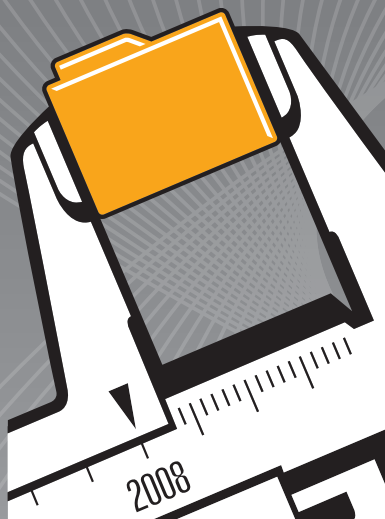


Материалы конференции/Conference Materials



# SEC(R) 2008

Software Engineering Conference (Russia)

*«Path to Competitive Advantage»*

Moscow, Russia  
October **21 – 25, 2008**



Microsoft



vmware



## Sponsors



Premium Sponsor

Intel, the world leader in silicon innovation, develops technologies, products and initiatives to continually advance how people work and live. Additional information about Intel is available at [www.intel.com/pressroom](http://www.intel.com/pressroom) and [blogs.intel.com](http://blogs.intel.com)



Premium Sponsor

INTSPEI (an abbreviation that stands for International Software & Productivity Engineering Institute) is a multinational research and development organization which was established to develop and distribute new Software Engineering and Productivity Engineering technologies to help information workers worldwide to become more efficient and productive by fully utilizing the power of today's and tomorrow's computers. For more details please visit us at: [www.intspei.com](http://www.intspei.com)



Gold Sponsor

Exigen Services is the leader in Outsourcing 2.0, the next generation of global application outsourcing. Exigen Services combines world class technical skills, recognized expertise in advanced development methodologies and industry experience to deliver results for global enterprises. Exigen Services is a pioneer in the use of distributed Agile methods for rapid and precise systems development for financial services, health care, telecom, and media and entertainment companies. Coupled with commercial terms which optimize financial alignment, Exigen Services reduces the risks and costs associated with solving enterprises' complex technical challenges.

Please find more at [www.exigenservices.com](http://www.exigenservices.com)



## Sponsors/Спонсоры



i n v e n t

Gold Sponsor

HP is a technology company that operates in more than 170 countries around the world. We explore how technology and services can help people and companies address their problems and challenges, and realize their possibilities, aspirations and dreams. We apply new thinking and ideas to create more simple, valuable and trusted experiences with technology, continuously improving the way our customers live and work.

No other company offers as complete a technology product portfolio as HP. We provide infrastructure and business offerings that span from handheld devices to some of the world's most powerful super-computer installations. We offer consumers a wide range of products and services from digital photography to digital entertainment and from computing to home printing. This comprehensive portfolio helps us match the right products, services and solutions to our customers' specific needs.



Gold Sponsor

VMware (NYSE: VMW) is the global leader in virtualization solutions from the desktop to the data center. Customers of all sizes rely on VMware to reduce capital and operating expenses, ensure business continuity, strengthen security and go green. With 2007 revenues of \$1.33 billion, more than 120,000 customers and nearly 18,000 partners, VMware is one of the fastest-growing public software companies. VMware is headquartered in Palo Alto, California, and on the Web at [www.vmware.com](http://www.vmware.com)

### VMware at a Glance

Founded: 1998; acquired by EMC in 2004; IPO in August 2007 (NYSE:VMW). Revenues: \$1.33 billion. Customers: 120,000, including 100% of Fortune 100. Employees: 6,300+. Headquarters: Palo Alto, CA, USA. Locations: 36 countries. Partnerships: Nearly 18,000

## Sponsors/Спонсоры

**Microsoft®**

Gold Sponsor

At Microsoft, we're motivated and inspired every day by how our customers use our software to find creative solutions to business problems, develop breakthrough ideas, and stay connected to what's most important to them. We are committed long term to the mission of helping our customers realize their full potential. Just as we constantly update and improve our products, we want to continually evolve our company to be in the best position to accelerate new technologies as they emerge and to better serve our customers.

Please find more at: [www.microsoft.com](http://www.microsoft.com)

**EMC<sup>2</sup>**  
where information lives®

General Sponsor

EMC Corporation (NYSE: EMC) is the world's leading developer and provider of information infrastructure technology and solutions that enable organizations of all sizes to transform the way they compete and create value from their information. We set standards in the field of data storage for all known computing platforms and provide storage for more than two thirds of the most important information in the world. EMC Corporation is represented in more than 50 countries and employs more than 30 000 worldwide.

Today EMC employs more than 200 software development and customer-facing professionals across operations in Russia. The St. Petersburg R&D center will be an integral part of EMC's global research and development network.

Please find more at [www.emc.com](http://www.emc.com)

**IBM®**

General Sponsor

IBM is the world's largest information technology company, with 90 years of leadership in helping businesses innovate. Drawing on resources from across IBM and key Business Partners, IBM offers a wide range of services, solutions and technologies that enable customers, large and small, to take full advantage of the new era of e-business. IBM Global Services is the world's largest information technology services and consulting provider.

For more information about IBM, visit <http://www.ibm.com>





# Adobe

General Sponsor

Adobe revolutionizes how the world engages with ideas and information. For 25 years, the company's award-winning software and technologies have re-defined business, entertainment, and personal communications by setting new standards for producing and delivering content that engages people virtually anywhere at anytime. From rich images in print, video, and film to dynamic digital content for a variety of media, the impact of Adobe solutions is evident across industries and felt by anyone who creates, views, and interacts with information. With a reputation for excellence and a portfolio of many of the most respected and recognizable software brands, Adobe is one of the world's largest and most diversified software companies.

Primary customer segments include:

- *Knowledge Workers and Enterprises:* Businesses and government organizations use Adobe desktop and server-based solutions to improve productivity, collaboration, and business processes inside and outside the organization.
- *Creatives and Designers:* With Adobe solutions, designers, publishers, photographers, and videographers are making brands and products stand out in crowded markets by designing compelling content for delivery in print, online, for video, and on mobile devices.
- *High-end Consumers:* A wide variety of enthusiasts use Adobe's popular solutions to develop, enhance, and deliver images and content in print and across a variety of digital devices.
- *Partners and Developers:* Adobe's technology platform enables developers, systems integrators, and software manufacturers to build dynamic applications that address business demand for improved interaction with information.

## Наши спонсоры



Premium Sponsor

Корпорация Intel, ведущий мировой производитель инновационных полупроводниковых компонентов, разрабатывает технологии, продукцию и инициативы, направленные на постоянное повышение качества жизни людей и совершенствование методов их работы.

Дополнительную информацию о корпорации Intel можно найти на Web-сайте [www.intel.com/pressroom](http://www.intel.com/pressroom), на русскоязычном Web-сервере компании Intel: [www.intel.ru](http://www.intel.ru), а также на сайте [blogs.intel.com](http://blogs.intel.com).



Premium Sponsor

Международный НИИ Проблем Программирования INTSPEI является исследовательской организацией, развивающей и внедряющей новые методологии разработки программного обеспечения. Продукты и услуги INTSPEI повышают эффективность команд, создающих ПО, позволяя им полностью использовать потенциал современных компьютерных технологий.

Подробная информация доступна на официальном сайте института: [www.intspei.com](http://www.intspei.com)



Gold Sponsor

Exigen Services — мультинациональная компания, занимающаяся разработкой заказного программного обеспечения и ставшая признанным экспертом в своей отрасли. Компания специализируется на разработке бизнес-приложений и web-решений разной сложности, осуществляет поддержку программных продуктов и систем, а также миграцию на новые технологические платформы, и оказывает услуги по заказному тестированию ПО. Компания входит в сотню ведущих американских поставщиков ИТ-услуг 2008 г.



## Sponsors/Спонсоры

Exigen Services активно использует гибкие методологии семейства Agile для оперативной и эффективной разработки промышленных систем. Среди клиентов компании крупнейшие мировые концерны из списка Fortune 500 и представители среднего бизнеса, такие, как Universal Music Group, Standard and Poor's, T-Mobile, AXA, Westpac Bank, Nationwide, British Telecom.

Штаб-квартира компании расположена в Сан-Франциско. У Exigen Services 18 офисов по всему миру, в том числе десять — в России, Белоруссии, Украине и странах Балтии. В петербургском Exigen Services сегодня трудится около 700 человек.

Доп. информация: [www.exigenservices.com](http://www.exigenservices.com)



**i n v e n t**

Gold Sponsor

НР — это компания из сферы высоких технологий, работающая в более чем 170 странах мира. Мы регулярно инвестируем в исследования, чтобы понять, как технологии и услуги помогают людям и компаниям решать назревшие проблемы, достигать желаемых результатов и реализовывать свои возможности и пожелания. Мы используем новые идеи и подходы для создания удобных в работе, более эффективных и надёжных технологий, чтобы с их помощью постоянно улучшать и делать удобнее жизнь всех наших клиентов — от отдельных пользователей до крупнейших корпораций.

Ни одна другая компания не может предложить такой широкий ассортимент технологических решений, как НР. Мы предлагаем продукцию для информационных инфраструктур и бизнеса, начиная от карманных устройств и заканчивая одними их самых мощных в мире суперкомпьютеров. Кроме того, мы предоставляем широкий спектр продуктов и услуг для потребителей, от цифровой фотографии и цифровых устройств для развлечения до компьютеров и устройств печати. Широкий ассортимент выпускаемой продукции позволяет предлагать технологии и услуги, наиболее полно соответствующие индивидуальным потребностям наших клиентов.



Gold Sponsor

Основанная в 1975 году, корпорация Microsoft является мировым лидером в производстве программного обеспечения, предоставлении услуг и разработке интернет-технологий для персональных компьютеров и серверов.

Корпорация Microsoft разрабатывает и выпускает широкий спектр программных продуктов. В их число входят настольные и сетевые операционные системы, серверные приложения для клиент-серверных сред, настольные бизнес-приложения и офисные приложения для пользователей, интерактивные программы и игры, средства для работы в сети интернет и инструменты разработки. Кроме того, Microsoft предлагает интерактивные (online) услуги, издает книги по компьютерной тематике, производит периферийное оборудование для компьютеров, занимается исследовательской деятельностью и разработкой новых компьютерных технологий.

Доп. информация: [www.microsoft.com](http://www.microsoft.com)

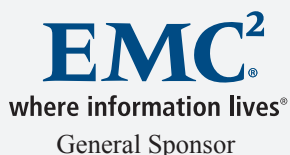


Gold Sponsor

Компания VMware (биржевой код NYSE: VMW) является мировым лидером в области решений виртуализации — от настольных систем до центров обработки данных. VMware помогает предприятиям любого размера снизить капитальные и эксплуатационные расходы, обеспечить непрерывность бизнеса, повысить уровень безопасности и защиту данных. С доходом в 1,33 миллиарда долл. за 2007, с числом заказчиков свыше 120 000 и числом партнеров более 18 000 VMware является одной из самых быстрорастущих открытых акционерных компаний-разработчиков программного обеспечения. Головной офис компании VMware находится в Пало-Альто (штат Калифорния), а веб-сайт — по адресу [www.vmware.com](http://www.vmware.com)



## Sponsors/Спонсоры



Корпорация EMC - мировой лидер на рынке продуктов, услуг и решений для хранения информации и управления ею. Мы задаем стандарты в области хранения данных для всех известных вычислительных платформ, и, благодаря нашим решениям, мы обеспечиваем хранение более чем двух третей самой важной информации в мире. Корпорация EMC представлена более чем в 50 странах. Персонал более 30000 человек.

Компания EMC работает на российском рынке с 2000 года. На сегодняшний день офисы EMC в Москве и Санкт-Петербурге насчитывают более 200 специалистов по разработке ПО и работе с клиентами. Санкт-Петербургский центр корпорации EMC по разработке программного обеспечения является частью глобальной программы EMC по инвестициям в исследования и разработки. Новый центр обеспечивает поддержку и осуществляет непрерывное развитие широкого спектра лидирующих на рынке программных продуктов.

Дополнительная информация: [www.russia.emc.com](http://www.russia.emc.com)



Adobe Systems Inc. в прошлом году отпраздновала свое 25-летие. За это время компания выпустила ряд продуктов, которые совершили революцию в подходе к работе с любыми видами контента – будь то текстовая информация или графические изображения, видео или веб.

Продукты Adobe стали стандартами де-факто для бизнеса:

- Создание стандарта PDF, без которого уже невозможно представить современный мир.
- Продукты для творческих профессионалов, ставшие легендами.

- Профильные решения для защиты информации и безопасного документооборота.

Adobe лидирует в каждой из отраслей, где представлены его продукты.



General Sponsor

Корпорация International Business Machines является одним из ведущих мировых поставщиков аппаратных и программных решений, лидером по разработке и производству информационных технологий и устройств хранения данных. Компания стремится лидировать в создании, разработке и производстве самых прогрессивных информационных технологий и воплощать их в решениях и услугах, оптимизирующих бизнес современной организации.



ТЕКАМА - Главный организатор SEC(R)2008, являющийся одним из ведущих учебных центров по ИТ и деловой эффективности с помощью ИТ. За 6 лет компания обучила более 15000 человек из более чем 3000 компаний в России, странах СНГ и Прибалтике. Преимуществом ТЕКАМА является наличие тесных связей с ключевыми игроками рынка ИТ, широкая и хорошо проработанная линейка продуктов, а также многолетнее партнерство с международным центром знаний в области ИТ – университетом Carnegie Mellon, единственным представителем которого ТЕКАМА является. Дополнительная информация: [www.tekama.com](http://www.tekama.com)



Высшая школа экономики учреждена 27 ноября 1992 г. Постановлением Правительства России. 3 филиала: в Санкт-Петербурге, Нижнем Новгороде, Перми. Система образования в ВШЭ позволяет обеспечить конкурентоспособность российского образования в самой его востребованной и одновременно самой слабой сфере – экономических и социальных науках.



Учебный Центр Luxoft - экспертное обучение по всем направлениям в области разработки ПО. В нашем центре можно пройти полный цикл обучения по всем ключевым направлениям в сфере разработки программного обеспечения.

Наши преподаватели (около 60-ти человек) - это ведущие эксперты компании Luxoft, работающие в реальных проектах и постоянно повышающие свою квалификацию.



conews

usethics

RUS®SOFT

softline®

IT-ИНФРАСТРУКТУРА БИЗНЕСА  
itexpert

iToday.ru

Agile Russia

IT.TUT.BY  
информационные технологии

новости информационных технологий  
itnews

developers.  
org.ua

PCWEEK  
RUSSIAN EDITION

INFOPARK

IT TOUR  
TRAVEL AGENCY

IT UKRAINE  
АССОЦИАЦИЯ

Открытые  
системы  
ЖУРНАЛ ДЛЯ АРХИТЕКТОРОВ ИНФОРМАЦИОННЫХ СИСТЕМ

itconf

INTSPEI P-Navigator:  
Smart Internet Search

USA MODEL  
LEAD

AGILE  
ukraine

it  
manager

BYTE  
РОССИЯ

АП.КИТ  
АССОЦИАЦИЯ ПРОГРАММИСТОВ  
И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

RSDN  
<http://www.rsdn.ru>

SCRUM  
guides



## Программный комитет/Program Committee

Vladimir Pavlov L. (INTSPEI) Co-chair of of Program Committee  
Antipov Ilya (RUSSEE Consulting) Co-chair of of Program Committee

- **Agamirzian Igor** (EMC<sup>2</sup>)
- **Avdoshin Sergey** (Higher School of Economics)
- **Babkin Alexander** (Motorola)
- **Barinov Alexey** (Telma)
- **Constantine Larry** (Constantine & Lockwood, Ltd.)
- **Doroshenko Anatoliy** (INTSPEI)
- **Gitsels Martin** (Siemens)
- **Gundarev Igor** (CBOSS)
- **Ivannikov Viktor** (ISP RAS)
- **Kemerer Chris** (University of Pittsburgh)
- **Khalin Dmitry** (Microsoft)
- **Komkov Alexey** (Cadence Design)
- **Kurbasov Michael** (CROC)
- **Larionova Zinaida** (IBA)
- **Marchuk Alexander** (IIS RAS)
- **Mellor Stephen** (StephenMellor.com)
- **Milov Denis** (TEKAMA)
- **Nesterov Vyacheslav** (EMC<sup>2</sup>)
- **Pankov Alexander** (Diasoft)
- **Paulk Mark** (Carnegie Mellon University)
- **Sabanin Boris** (Intel)
- **Sambuk Alexander** (LUXOFT)
- **Sutherland Jeff** (Scrum, Inc)
- **Taran Gil** (Carnegie Mellon University)
- **Tarasenko Alexander** (Miratech)
- **Telnov Yuriy** (MESI)
- **Terekhov Andrey A.** (Microsoft)
- **Terekhov Andrey N.** (Saint Petersburg State University)



Реализуйте потенциал своего ПО

## РЕШЕНИЯ INTEL® ДЛЯ РАЗРАБОТЧИКОВ ПО

Создавайте приложения с помощью инструментов, в основе которых лежит наше знание аппаратного обеспечения:

[www.intel.com/software/products](http://www.intel.com/software/products)

- **Intel® Compilers** – Повысьте производительность своего ПО за счет использования оптимизирующих компиляторов C++ и Fortran. Удобный отладчик полностью поддерживает параллельные приложения
- **Intel® Performance Libraries** – Постройте высокопроизводительные многопоточные математические, мультимедийные и другие приложения, используя базовые блоки Intel® Performance Primitives и Intel® Math Kernel Library
- **Intel® VTune™ Performance Analyzer** – Сделайте тюнинг приложений проще и эффективнее с помощью удобного графического интерфейса пользователя и без перекомпиляции
- **Intel® Threading Analysis Tools** – Воспользуйтесь инструментами Intel® Thread Checker и Intel® Thread Profiler, а также конструктивными компонентами Intel® Threading Building Blocks в процессе разработки ПО. Оптимизируйте производительность и убедитесь в корректности своих многопоточных приложений
- **Intel® Cluster Tools** – Базируйте разработку своего высокопроизводительного ПО для кластерных систем на специализированных инструментах Intel, поддерживающих полный цикл разработки распределенных приложений
- **Intel® SOA Expressway** – Применяйте высокопроизводительное программное решение для преобразования и обмена данными в информационных бизнес-сетях
- **Intel® XML Software Products** – Достигайте максимальной эффективности SOA и XML приложений

## ПОДДЕРЖКА INTEL® ДЛЯ РАЗРАБОТЧИКОВ ПО

- Общайтесь с разработчиками и инженерами Intel и узнавайте факты, новости и мнения из первых рук через **Intel® Software Network**: [software.intel.com](http://software.intel.com)
- Разрабатывайте коммерческие приложения, используя целый комплекс преимуществ, направленных на их поддержку в течение всего цикла создания продукта и предоставляемых **Intel® Software Partner Program**: [www.intel.com/partner](http://www.intel.com/partner)

# Scaling-up Agility to Globally Distributed Teams – The Eclipse Way

Erich Gamma  
IBM

## Abstract



Many aspects of agile development support smaller teams working together in proximity. The Eclipse Platform was developed using agile practices, but the team never had the luxury of developing in a single location or even a single time zone. And, in the course of pursuing our goal of ongoing project health through continuous consumption and feedback, we adapted and evolved our practices and those we learned from the agile community. This has resulted in what we now refer to as the "Eclipse Way" practices that we (and others) have applied to other software development efforts as well. In this workshop we take a look at the Eclipse Way with a particular focus on how we tuned our agile practices to work for a globally distributed team.

**Bio:** Erich Gamma is a Distinguished Engineer at IBM Rational Software's Zurich lab. He is the technical lead of the Jazz project and Rational Team Concert. He was the original lead of the Eclipse Java development environment and was on the Project Management Committee for the Eclipse project.

Erich is a member of the Gang of Four, which is known for its classical book, *Design Patterns - Elements of Reusable Object-Oriented Software*. Erich has collaborated with Kent Beck on developing JUnit, the de facto standard testing tool for Java software, and on writing the book contributing to Eclipse: *Principles, Patterns, and Plug-ins*.

## **Путь Eclipse: Масштабирование гибких методов (Agile) для распределенных команд**

**Эрих Гамма  
IBM**

### **Тезисы**

Многие практики гибких методов разработки хороши для маленьких команд, работающих вместе, в непосредственной близости друг от друга. Платформа Eclipse была разработана с использованием гибких практик, но наша команда никогда не имела такой роскоши, как работа в одном месте или даже в одном часовом поясе. В ходе проекта, по мере достижения нашей цели, мы непрерывно применяли и совершенствовали те практики и методы, которые обычно применяют в гибких процессах. Это привело к появлению подходов к разработке, которые мы называем «путь Eclipse». В этом семинаре мы посмотрим на «путь Eclipse» под определенным углом, а именно: как применить практики гибких процессов для работы в большой распределенной команде.

Биография: Эрих Гамма является ведущим инженером Цюрихской лаборатории IBM Rational Software, где он осуществляет техническое руководство проектом Jazz. Эрих был первоначальным руководителем проекта Eclipse. Он получил широкую известность в качестве одного из четырех авторов классической книги по шаблонам проектирования «Design Patterns - Elements of Reusable Object-Oriented Software», которая выдержала более 35 изданий суммарным тиражом более 500000 экземпляров. Кроме этого, совместно с Кентом Бекон Эрик стоял у истоков JUnit, ставшего сегодня стандартным инструментом тестирования Java-программ, и является соавтором книги об Eclipse.

# AGILE MDA

Stephen Mellor

## Abstract

"Agile" and "MDA" (Model-Driven Architecture) do not often appear in the same sentence. Agility promotes the notion that testing a running system is better than building descriptions of them, which appears to speak against the construction of models, surely the heart of model-driven architecture. However, models can now be executed, and many of the principles of agility can be applied to executable models in the context of MDA. This synthesis of executable models and an agile process can change fundamentally how we build software.

MDA (model-driven architecture) is a broad church covering a number of different approaches to model-driven development. Most commonly, people think of models as blueprints that are filled in with code, and model-driven development supporting automation of transformations between these several models. That is, MDA is commonly viewed as supporting "heavyweight" process-heavy modeling techniques; but MDA can do better than this.

Agile MDA is based on the notion that code and executable models are operationally the same. Hence, the principles of the Agile Alliance testing first, immediate execution, racing down the chain from analysis to implementation in short cycles, for example can be applied equally to models. An executable model, because it is executable, can be constructed, run, tested and modified in short incremental, iterative cycles.

To reach this happy state, models must be complete enough that they can be executed standing alone. There are no "analysis" or "design" models, because all models are equal. Models are linked together, rather than transformed, and they are then all mapped to a single combined model that is then translated into code according to a single system architecture. This approach to MDA is called Agile MDA.



**Bio:** Stephen J. Mellor is an internationally recognized pioneer in creating effective, engineering approaches to software development. In 1985, he published the widely read Ward-Mellor trilogy

Structured Development for Real-Time Systems and in 1988, the first books defining object-oriented analysis.

Stephen also published Executable UML: A Foundation for Model-Driven Architecture in 2002. His latest book MDA Distilled: Principles of Model-Driven Architecture was published in 2004. He is active in the Object Management Group, chairing the consortium that added executable actions to the UML, and he is presently working on a standard for executable UML. He was a two-term member of the OMG Architecture Board and active in specifying MDA.

He is a signatory to the Agile Manifesto. In his copious spare time, he acts as chair of the IEEE Software Advisory Board. Stephen was until recently chief scientist of the Embedded Software Division at Mentor Graphics.



# AGILE MDA

Стив Меллор

## Тезисы

"Agile" и "MDA" (Model-Driven Architecture) не часто появляются в одном и том же предложении. Гибкие методы поддерживает идею, что тестирование работающей системы лучше, чем создание ее описаний. Эта идея, казалось бы, противоречит концепции построения моделей перед кодированием, безусловно являющейся основой подхода Model-Driven Architecture. Тем не менее, модели теперь могут быть реализованы, и многие из принципов гибких методов могут быть применены к реализуемым моделям в контексте МДА. Синтез исполняемых моделей и гибкого процесса может коренным образом изменить наш подход к созданию программного обеспечения.

MDA (model-driven architecture) это широкая концепция, охватывающая различные модельно-ориентированные подходы к разработке программного обеспечения. Чаще всего, люди думают о моделях, как о чертежах или схемах, которые потом преобразуются в код, и МДА поддерживает автоматизацию перехода от модели к коду. Таким образом, MDA, обычно рассматривается как подход, поддерживающий "тяжеловесные" техники моделирования, но на самом деле МДА – это нечто большее.

Подход Agile MDA основан на идее, что исполняемый код и модели это практически одно и то же. Таким образом, принципы Agile - тестирование в первую очередь, немедленное исполнение, быстрый переход от анализа к реализации, короткие циклы - могут быть применены в равной степени к моделям. Исполняемая модель, именно потому что она является исполняемой, может быть построена, запущена, протестирована и изменена во время коротких, итерационных циклов.

Чтобы достичь этого прекрасного состояния, модели должны быть достаточно полны, чтобы быть исполненными сами по себе. При этом не должно быть отдельных моделей "анализа" или "дизайна", потому что все модели одинаковы. Модели скорее связаны между собой, чем преобразованы одна из другой, и впоследствии они все преобразуются в одну общую

модель, которая затем переводится в код в соответствии с единой архитектуры системы. Такой подход к MDA называется Agile MDA.

**Биография:** Стив Меллор (Stephen Mellor) - всемирно признанный профессионал и пионер в создании эффективных инженерных подходов в области разработки программного обеспечения. В 1985 он опубликовал широко известную во всем мире книгу в трех частях: «Структурированное проектирование систем реального времени» (Ward-Mellor trilogy Structured Development for Real-Time Systems),- а затем в 1988 и первые книги, посвященные объектно-ориентированному анализу.

В 2002 году Стив опубликовал книгу «Executable UML: A Foundation for Model-Driven Architecture» (UML как основа MDA (Model-Driven Architecture)). Его наиболее позднее издание, посвященное MDA: «Principles of Model-Driven Architecture » (Принципы MDA),- опубликовано в 2004 году. Он принимает активное участие в OMG (Object Management Group), возглавляя консорциум, который добавляет новые функции в UML, а в настоящее время он работает над стандартом для UML. Два срока подряд он был членом архитектурного совета OMG (Object Management Group) и специализировался по вопросам MDA (Model-Driven Architecture).

Стив, один из тех, кто подписал Agile Manifesto. Появление этого документа ознаменовало начало новой эры в разработке программного обеспечения. На постоянной основе Стив возглавляет консультационный совет по программному обеспечению при IEEE. Стив так же был руководителем исследовательских работ подразделения программного обеспечения корпорации Mentor Graphics (лидирующая корпорация в области проектирования и автоматизации).

# SDLC Fine-Tuning

## Optimize Your Software Development Life Cycle!

This specialized consulting service is intended to help software companies:

- Improve their efficiency;
- Enable engineers to eliminate bugs early;
- Shorten overall development time;
- Improve software quality.



## Our SDLC Fine-Tuning Service Includes Three Major Phases:

- Identify bottlenecks in the software development process currently being used;
- Design and implement process improvements addressing identified issues;
- Teach software engineers how to effectively implement and use these process changes.

Please write us at [info@intspei.com](mailto:info@intspei.com) to request the service or to learn more about INTSPEI solutions.



International Software and Productivity Engineering Institute  
1979 Marcus Avenue, Suite 210, Lake Success, NY 11042, USA  
Call: +1-877-INTSPEI (+1-877-468-7734)  
Visit: [www.intspei.com](http://www.intspei.com)



# Technical evangelism and its place in software development process

Aleksandr Lozhechkin  
Department of Platform and Evangelism  
Microsoft

## Abstract

Lately the word “evangelist” which is traditionally related to religion, has begun to be used more and more often in a new context, as an evangelist of ideas, and ideas which are far away from religion. For instance, Richard Stollman is often called an Open Source evangelist; official position of Anton Nossik in SUP is again an evangelist. Russian Microsoft subsidiary, which, by the way, introduced this term in the new context in Russia, has a special division of technical evangelists. The goal of this session is to describe what evangelism really means, how beneficial it can be for a software development company, what kind of projects are most suitable to be evangelized and how to effectively and efficiently integrate evangelism efforts to the software development process



**Bio:** Alexander Lozhechkin leads a team of technical evangelists in the Department of Platform and Evangelism group in Microsoft Russia starting from January 2007. Before this Alexander had been an Architect and Developer Evangelist in Microsoft since 2004. Before Microsoft

Alexander was leading software development team of document management solution DocsVision in Digital Design company. Alexander holds master's degree in Electronics from Saint-Petersburg State University of Aerospace Instrumentation, PhD in Computer Science from University of Railways Communications.

# **Технический евангелизм и его место в процессе создания программного обеспечения**

**Александр Ложечкин**  
**Департаменте стратегических технологий**  
**ООО «Майкрософт Рус»**

## **Тезисы**

В последнее время слово «евангелист», традиционно относящееся к области религии, стало все чаще и чаще употребляться в новом для себя качестве - евангелиста идей, причем находящихся весьма далеко от религии. Например, Ричарда Столлмана часто называют евангелистом Open Source, должность Антона Носсика в СУПе тоже называется евангелист. А в Российском представительстве Microsoft, с которого, кстати, началось проникновение этого слова в новое для себя качество в Россию, евангелистов вообще целый отдел. Целью доклада является рассказ о том, что же такое евангелизм на самом деле, какую пользу евангелизм может принести технологической компании, в какого рода проектах требуется целенаправленно заниматься евангелизмом, а также как эффективно и не тратя избыточных ресурсов интегрировать евангелистскую деятельность в процесс создания программного обеспечения.

**Биография:** Александр Ложечкин возглавляет Группу евангелистов в Департаменте стратегических технологий ООО «Майкрософт Рус» с января 2007 года. До прихода на текущую позицию, Александр Ложечкин работал евангелистом с 2004 года. До прихода в Microsoft Александр возглавлял группу разработки системы документооборота DocsVision в компании Digital Design. В 2000 году Александр Ложечкин окончил факультет радиотехники, электроники и связи Санкт-Петербургского государственного университета аэрокосмического приборостроения, в 2004 году защитил диссертацию в Петербургском государственном университете путей сообщения.

# **The economics of the staff training at the enterprise**

**Andrey N. Terekhov**  
**Lanit-Tercom Inc**

**Karina Terekhova**  
**NOKIA**

## **Abstract**

It is well-known that the main problem of the modern IT industry is the severe shortage of staff. They teach IT experts at classical universities and technical colleges, in colleges (technical schools), on the paid short-term courses, an idea of the second or additional higher education becomes more and more popular, but the lack of qualified staff doesn't reduce. Many large IT companies open training-centers where they give additional knowledge or even retrain the staff, found by their HR. Such additional costs essentially increase a firm overhead charge, but often it is the only way to solve the problem. Sometimes it's necessary only to explain to the new employee existing company standards and accepted technologies, but if it is necessary to learn programming languages and effective algorithms then it is hardly to understand what is the real alma mater of employee: his college or his company.

Labour market "overheat" automatically causes unrestrained growth of salaries. There are problems for employers as a result. The social tax in Russia is high enough – 26,2 % from the salary (in the western countries it does not exceed 15 %, and often it is paid by the worker, instead of the employer).

Rent and communication costs grow every year – or at least don't fall. Therefore the maintenance of the highly skilled staff costs much to the enterprise. Generally nobody can start work constructively immediately, so it causes additional essential losses for the company.

Every year about 150 thousand IT of experts graduate from high schools in Russia– one of the best values in the world, but the lack of them is still huge. Besides, training level is different in different universities and colleges, especially in small towns. One more problem – high schools practically do not train students in many important skills for the industry, for example, team work, version control, culture of reports, budgeting, methods of quality assurance and so on.

Some colleges begin to work in another erroneous manner – become to be specialized courses of one certain technology, for example, SAP or.NET, forgetting about basic knowledge.

Thus, any IT company should spend rather big amount of money for their staff's training. In this report we will describe some variants of this activity and will try to compare them from the economic point of view.



**Bio:** 1971 - graduated from the Mathematics and Mechanics Faculty of Leningrad State University. Received a diploma with honour in the field of Computer Science.

1978 - Ph.D. from Leningrad State University. Thesis: "Synthesis of efficient

programs".

1991 - Professorship, habilitation thesis: "Software technology for real-time embedded systems". In the same year 1991 founded and became director of State Enterprise "Tercom".

In year 1996 founded and headed Software Engineering Chair of St. Petersburg State University. In year 1998 founded and became CEO of the "Lanit-Tercom Inc". In 2008 "Lanit-Tercom Inc" and Moscow software company "Artezio" merged into AT Software. Andrey N. Terekhov heads Board of Directors and supervises R&D department of the new company.

In 1999 Andrey N. Terekhov was one of founders of software developers association of St.-Petersburg – Fort-Ross. When in 2004 on the base of Fort-Ross the All-Russia association of software developers – RUSSOFT has been created, Andrey N. Terekhov has been selected as its first Chairman of board of directors.

Professor Andrey N. Terekhov has more than 70 scientific publications (including 4 books). He is a member of ACM and IEEE.



# **Экономика подготовки кадров на предприятии**

**Андрей Н. Терехов**  
**ЗАО «Ланит-Терком»**

**Карина Терехова**  
**NOKIA**

## **Тезисы**

Общеизвестно, что основной проблемой современной ИТ индустрии является острый дефицит кадров. ИТ специалистов готовят в классических университетах и технических вузах, в колледжах (техникумах), на платных краткосрочных курсах, всё более популярной становится идея второго или дополнительного высшего образования, но квалифицированных кадров все равно не хватает. Многие крупные ИТ компании открывают свои тренинг-центры, в которых доучивают или даже переучивают специалистов, найденных их HR службами. Такие дополнительные траты существенно увеличивают накладные расходы предприятия, но часто другого пути просто нет.

Хорошо, если объяснять новому сотруднику надо только существующие на предприятии стандарты и принятые технологии, но если приходится учить языкам программирования и эффективным алгоритмам, тогда вообще непонятно, кто чем занимается.

«Перегретость» рынка труда автоматически влечет безудержный рост зарплат. В связи с этим возникают проблемы у работодателей. В России довольно высокий единый социальный налог (ЕСН) – 26,2% от зарплаты (в западных странах он не превышает 15%, а часто его вообще платит работник, а не работодатель). Каждый год растет стоимость аренды площадей, а стоимость коммуникаций – никак не падает. Как следствие, содержание высококвалифицированного специалиста дорого обходится предприятию. Если же он не может немедленно приступить к созидательной работе, а чаще всего, это именно так, предприятие несет дополнительные существенные потери.

Вузы России готовят порядка 150 тысяч ИТ специалистов в год – один из крупнейших показателей в мире, но потребность в них еще выше, кроме того, уровень подготовки сильно разнится от вуза к вузу, особенно в небольших городах. Ещё одна проблема – вузы практически не обучают многим важным для

промышленности навыкам, например, работе в команде, версионному контролю, культуре отчетов, бюджетированию, методам обеспечения качества и так далее. Некоторые вузы скатываются в другую крайность – превращаются в курсы по подготовке в рамках ровно одной технологии, например, SAP или .NET, забывая о фундаментальной подготовке.

Таким образом, каждое IT предприятие в любом случае должно тратить довольно большие деньги на подготовку кадров. В данном докладе мы опишем некоторые варианты этой деятельности и попробуем сравнить их с экономической точки зрения.

**Биография:** Андрей Н. Терехов закончил с отличием математико-механический факультет ЛГУ в 1971 году, был первым выпускником новой кафедры математического обеспечения ЭВМ. В 1978 Андрей Н. Терехов защитил кандидатскую диссертацию (к.ф.-м.н.) по технике трансляции, а в 1991 докторскую диссертацию (также физ-мат наук) по технологии программирования встроенных систем реального времени. В 1994 году стал профессором.

В 1976 году Андрей Н. Терехов был назначен и.о. заведующего лабораторией системного программирования НИИММ ЛГУ, в 1984 – стал официальным заведующим этой лаборатории, в 1987 им был образован совместный комплексный отдел Минпромсвязи СССР и Минобр РСФСР, на основе которого в 1989 году был образован НИИ «Звезда» ЛНПО «Красная Заря», в котором Андрей Н. Терехов работал заместителем директора по перспективным научным исследованиям (по совместительству). В 1996 году Андрей Н. Терехов создал и возглавил кафедру системного программирования СПбГУ, а в 2002 году – создал и возглавил НИИ информационных технологий СПбГУ. Имеет более 70 научных публикаций (из них 4 книги).

В 1991 году Андрей Н. Терехов создал ГУП «Терком», а в 1998 году – в дополнение к ГУП «Терком» было создано ЗАО «Ланит-Терком», в которых Андрей Н. Терехов был генеральным директором. В 2008 году ЗАО «Ланит-Терком» объединилось с московской компанией «Артезио», во вновь образованном предприятии АТ Software Андрей Н. Терехов – возглавляет Совет директоров и руководит R&D департаментом.

В 1999 году Андрей Н. Терехов явился одним из учредителей ассоциации разработчиков ПО Санкт-Петербурга – Форт-Росс. Когда в 2004 году на базе Форт-Росса была создана

Всероссийская ассоциация разработчиков ПО – РУССОФТ, Андрей Н. Терехов был избран её первым Председателем Совета директоров.

Андрей Н. Терехов – член АСМ и IEEE.

## **Вступление**

Основной проблемой современной ИТ индустрии является острый дефицит кадров. ИТ специалистов готовят в классических университетах и технических вузах, в колледжах (техникумах), на платных краткосрочных курсах, всё более популярной становится идея второго или дополнительного высшего образования, но квалифицированных кадров все равно не хватает. Многие крупные ИТ компании открывают свои тренинг-центры, в которых доучивают или даже переучивают специалистов, найденных их HR службами. Такие дополнительные траты существенно увеличивают накладные расходы предприятия, но часто другого пути просто нет.

Хорошо, если объяснять новому сотруднику надо только существующие на предприятии рабочие процессы и принятые технологии, но если приходится учить языкам программирования и эффективным алгоритмам, тогда его ценность для предприятия резко падает, и предприятию пора пересмотреть политику набора кадров.

10-15 лет назад нехватку кадров списывали на «утечку мозгов», когда лучшие и самые инициативные специалисты уезжали на Запад. Сейчас этот процесс почти полностью прекратился. Более того, сейчас на многих предприятиях Санкт-Петербурга работают выходцы как из ближнего, так и дальнего зарубежья. Кадров все равно не хватает.

«Перегретость» рынка труда автоматически влечет безудержный рост зарплат. Зачем уезжать в Америку, если можно перебежать на соседнее предприятие, существенно повысив свою зарплату. При этом ты живешь на Родине, вокруг все говорят на родном языке, подоходный налог (13%) один из самых низких в мире, а цены на жилье и транспорт хоть и быстро растут, но все еще в 2-3 раза ниже, чем в Америке или Европе.

Проблемы возникают у работодателей. В России довольно высокий единый социальный налог (ЕСН) – 26,2% от зарплаты (в западных странах он не превышает 15%, а часто его вообще платит работник, а не работодатель). Каждый год растет стоимость аренды площадей, а стоимость коммуникаций, вопреки прогнозам, не падает. Как следствие,

содержание высококвалифицированного специалиста дорого обходится предприятию. Если же он не может немедленно приступить к созидательной работе, а чаще всего, это именно так, предприятие несет существенные потери.

Вузы России выпускают около 150 тысяч IT специалистов в год. Это один из крупнейших показателей в мире, но потребность в них еще выше, кроме того, уровень подготовки сильно разнится от вуза к вузу, особенно в небольших городах. Ещё одна проблема – вузы практически не обучают многим важным для промышленности навыкам, например, работе в команде, основам управления проектами, версионному контролю, культуре отчетов, бюджетированию, методам обеспечения качества и так далее. Некоторые вузы скатываются в другую крайность – превращаются в курсы по подготовке в рамках одной технологии, например, SAP или .NET, забывая о фундаментальной подготовке.

Таким образом, каждое IT предприятие в любом случае должно тратить довольно большие деньги на поиск и переподготовку кадров. В данной статье мы опишем некоторые варианты этой деятельности и попробуем сравнить их с экономической точки зрения.

## **1. Прибыльность бизнеса разработки ПО**

Каждому предприятию нужно много квалифицированных младших программистов и тестеров (не менее половины всех разработчиков). Именно эти категории работников чаще всего подвержены текучке и они же требуют дополнительного образования. Во сколько такой специалист обходится предприятию?

Пусть его ежемесячная зарплата составляет \$1000. Это не слишком большая зарплата, но не забывайте, что речь идет о младших программистах и тестерах. Тогда 26,2% предприятие должно уплатить в качестве единого социального налога (ЕСН), то есть \$262. Только что в России ввели новый закон, по которому некоторые категории IT предприятий (например, экспортеры ПО) могут платить ЕСН по ставке 14-16%, но широкой практики его применения ещё нет.

Специалиста надо разместить в хорошо оборудованном помещении, иначе трудно будет найти на рынке труда кандидата, да и заказчики обращают большое внимание на место работы. В настоящее время (лето 2008 года) в Санкт-Петербурге стоимость квадратного метра в бизнес-центре среднего класса составляет \$35. По российским нормам на

одного сотрудника должно приходиться в среднем 10 м<sup>2</sup> (включая чайные комнаты, переговорные, туалеты и т.п.). Таким образом, аренда составляет \$350 на одного специалиста.

Накладные расходы (расходы на административную, связь, обслуживание техники и т.п.) обычно составляют 30% от зарплаты. Вообще-то, аренда помещений включается в накладные расходы, но мы её выделим, поскольку в Санкт-Петербурге цена аренды быстро растет. Отнесем \$300 на накладные расходы (кроме аренды).

Предприятие должно работать с прибылью, причем, если прибыль составляет менее 10% от оборота, налоговые органы начинают подозревать предприятие в уклонении от уплаты налогов. Поэтому оценим прибыль в 20% от зарплаты (что дает примерно 10% от оборота), то есть \$200, с которой удерживается налог на прибыль (24%), т.е. \$48

Таким образом, суммарные затраты на 1 младшего программиста или тестера составляют

$$\$1000 + \$262 + \$350 + \$300 + \$48 = \$1960 \text{ (в месяц)}$$

Услуги молодого специалиста оцениваются заказчиками не более \$15/час, т.е. \$2400/месяц (в западных компаниях принято считать, что в месяце 160 рабочих часов). Это не зарплата, а именно общая оценка, что может получить предприятие, предоставляя услуги данного специалиста.

К сожалению, редко когда удастся обеспечить загрузку специалиста более, чем на 85% (болезни, перерывы в заказах, обучение). Поэтому выручка на одного молодого специалиста оценивается так:

$$\$2400 * 0,85 = \$2040$$

Как видим, получается не слишком большая прибыль (4%, даже меньше расчетных \$200). Для увеличения прибыли необходимо продавать услуги даже молодых программистов дороже \$15/час, а это возможно только путем повышения их квалификации. Поэтому удержание кадров, уже работающих на предприятии, важнее политики приобретения новых кадров. Рассмотрим варианты поиска и подготовки кадров и оценим их также по продолжительности работы на предприятии найденных кадров.

## 2. Оценка начальных затрат на поиск специалиста на рынке

В Санкт-Петербурге 25 вузов ежегодно готовят около 4000 выпускников по основному списку специальностей информационно-коммуникационных технологий (ИКТ). Потребность нашего города в ИКТ-специалистах оценивается в десять раз больше. 5-10 лет назад специалистов по ИКТ готовили существенно меньше, зато большое количество инженеров других специальностей, не нашедших работу в соответствии со своим образованием, переучивались на программистов. Сейчас в России наблюдается подъем промышленности, поэтому этот процесс практически полностью прекратился.

В крупные предприятия выпускники приходят и сами, но все-таки даже известные предприятия вынуждены обращаться в рекрутинговые агентства. За каждого найденного специалиста агентство требует 15-20% от его годовой зарплаты, т.е. в наших оценках – \$3-4 тысячи.

Умение программировать на нужном для данного заказа языке обычно проверяется уже на собеседовании при приеме на работу. Но ведь нужно научить принятым в данном коллективе средствам версионного контроля, автоматического тестирования, формам еженедельной и итоговой отчетности, стандартам на представление исходных текстов программ и т.д. Нужно, чтобы новый специалист познакомился с теми частями кода, которые были написаны до него, но которые ему (ей) предстоит использовать. Нужно разобраться с трудностями предметной области, изучить новые для него протоколы. Список необходимого дообучения всегда разный и всегда большой.

По разным оценкам, чтобы новый специалист начал успешно работать, надо потратить на его дообучение 1.5 – 3 месяца, то есть три-шесть тысяч долларов. Эта оценка является нижней, так как иногда приходится посылать молодого специалиста на какие-нибудь курсы (еще \$1-2 тысячи), на новичка тратят время другие сотрудники (также оцениваем в \$1-2 тысячи), иногда процесс занимает и более 3 месяцев. И уж совсем трудно оценить тот факт, что, пользуясь «перегретостью» рынка труда, некоторые специалисты (а их становится всё больше) взяли себе за правило менять место работы каждые полгода-год, повышая свою зарплату, но предприятие несет при этом большие потери. Тем не менее

получаем оценку на начальные затраты в \$8-14 тысяч на одного сотрудника-новичка.

В нормальной ситуации после участия в 2-3 успешных проектах (1-2 года) младший программист или тестер повышается в должности, а на его место надо искать нового специалиста. Следовательно, начальные затраты на нового младшего программиста или тестера надо окупить максимум за два года.

При норме прибыли на одного сотрудника в 4% (как вычислено выше) и средней месячной выручке \$2040, годовая прибыль на одного сотрудника такого типа не превышает \$1000 ( $\$2040 * 12 * 0.04$ ). Из этих расчетов видно, что окупить даже минимальные расходы на рекрутинг «с нуля» за два года не удастся.

Встает вопрос – почему предприятия идут на заведомо непродуктивные расходы? Во-первых, ради поддержания объемов производства, так как для получения крупных заказов и увеличения размаха операций необходимо доказать свои возможности к расширению в случае необходимости.

Во-вторых, мы возвращаемся к коэффициенту текучки кадров. Если он невелик, то в условиях крупного предприятия есть расчет на будущее повышение прибыльности данного сотрудника. Что не отменяет необходимость ежегодно повторять процесс набора специалистов начального уровня наиболее экономичным способом.

### **3. Как мы готовим кадры для себя**

В предыдущих разделах статьи мы показали, как трудно предприятию найти программистов на рынке труда и сколько это примерно стоит. А есть ли другие способы решения этой проблемы?

Поскольку один из авторов статьи не только руководит предприятием, но и является заведующим кафедрой системного программирования СПбГУ, идея привлечь программистские предприятия к процессу подготовки кадров в университете ему пришла в голову много лет назад.

Разумеется, эта идея не нова. Еще в СССР отлично работали базовые кафедры крупных предприятий в известных вузах. Но та экономика не была рыночной, никто не считал, во сколько обходится предприятию каждый подготовленный специалист. Надо – и всё! В наше время многие крупные фирмы оказывают поддержку университетам в форме



стипендий студентам и преподавателям, оплаты командировок на свои научно-технические конференции, бесплатного ПО и своего оборудования. Это всё важно, но не носит системного характера и не может кардинально изменить качество подготовки студентов, а главное- их мотивацию при выборе первого места работы.

Нам известно несколько случаев, когда предприятие в сотрудничестве с университетом бралось организовать полноценное обучение в качестве второго высшего образования. Почти всегда это было платное обучение, т.е. предприятие пыталось не только готовить для себя кадры, но и организовать новый бизнес. Все эти попытки не вышли за рамки эксперимента в силу экономических причин.

Таким образом, простая и понятная идея подготовки кадров предприятием в сотрудничестве с университетом не так просто реализуема. В частности, когда много лет назад А.Н. Терехов стал продвигать эту идею в ЗАО «Ланит-Терком»<sup>1</sup>, где он работал генеральным директором, директора департаментов приняли её в штыки. «Профессор, Вы пытаетесь решить проблемы своей кафедры за счёт предприятия. С чего это мы должны заставлять своих сотрудников выполнять работу преподавателей кафедры?» Но многомесячные потери из-за того, что наша служба HR при всём старании не могла найти подходящие кадры в достаточных количествах, изменила ситуацию. Методом проб и ошибок, а также благодаря усилиям отдельных энтузиастов нам удалось выстроить методику подготовки студентов, обеспечивающую приток 20-25 хороших специалистов ежегодно, при затратах, меньших, чем упомянутые в параграфах 1-2 данной статьи на поиск специалистов «с улицы». Самое главное, такой подход хорошо работает не только в подготовке кадров, но и в их удержании на предприятии.

Здесь мы попытаемся сформулировать основные моменты нашей методики, основанные на многолетнем успешном сотрудничестве ЗАО «Ланит-Терком» и математикомеханического факультета СПбГУ. Мы считаем, что при соблюдении перечисленных далее требований, этот опыт может быть тиражирован и другими предприятиями.

---

<sup>1</sup> Летом 2008 года ЗАО «Ланит-Терком» было преобразовано в компанию AT-Software путем слияния с московской компанией Artezio.

Во-первых, университет-партнер должен обеспечивать хорошую базовую фундаментальную подготовку по математическому анализу, алгебре, геометрии, дискретной математике, эффективным алгоритмам и другим разделам современной математики. Никакое предприятие не сумеет этого обеспечить, но эти знания абсолютно необходимы программным инженерам.

Во-вторых, университет должен следовать международным образовательным стандартам Computer Science Curricula 2001 и Software Engineering Curricula 2004. Российские университеты традиционно сильны в математике и «классическом» программировании, но вопросы управления проектами, обеспечения качества, коллективной разработки часто остаются за рамками учебной программы. Упомянутые выше международные стандарты восполняют эти проблемы. Для мат-меха СПбГУ первое требование выполнялось во все периоды существования факультета, а второе требование привело к появлению нескольких дополнительных курсов. Сотрудники кафедры приняли участие в переводе этих стандартов на русский язык.

В чем же тогда роль предприятия? И российские, и международные стандарты требуют проведения ежегодной научной работы студентов (курсовая работа), прохождения преддипломной практики на производстве, наконец, выполнения заключительной дипломной работы, причем стандарт по программной инженерии настойчиво рекомендует именно коллективные работы. Предприятие может выделить научных руководителей-тьюторов для руководства небольшими группами студентов по темам, интересным для предприятия. Предприятие может предоставить компьютеры и другие ресурсы, необходимые для выполнения работы.

Предприятие может обеспечить необходимую университету обратную связь, т.е. аргументированную информацию, что хорошо, а что плохо в образовательном процессе университета. Наконец, предприятие может предоставить преподавателей по новейшим технологиям, с которыми кадровые преподаватели ещё не успели познакомиться.

Таким образом, третьим требованием является готовность предприятия взять на себя перечисленные обязательства перед университетом, при этом перед предприятием встают две основные проблемы (кроме, разумеется, материальных затрат):

1. Близость научных и технологических направлений. Ни одна кафедра, ни один факультет не может быть одинаково силен сразу во всех направлениях ИТ. Предприятию нужно стремиться выбирать университеты-партнеры со сходными основными интересами.

2. Гарантия инвестиций. Предприятие вкладывает в подготовку деньги, усилия своих сотрудников, другие ресурсы, а студент, получивший дополнительную подготовку, скажет спасибо и уйдет на другое предприятие. Эта проблема родственна уже упоминавшейся важной проблеме удержания кадров и решается схожими методами.

Наш многолетний опыт подсказывает, что если уж студент работал над курсовой работой 3 и 4 курса на предприятии, прошел там преддипломную практику и написал там же дипломную работу, он уже хорошо освоился на предприятии, знает условия труда и научно-технические направления предприятия, которые интересны ему, знает лично многих сотрудников, то он (она) будет работать на предприятии много лет, в среднем дольше сотрудников, найденных «с улицы».

В среднем, сотрудник, найденный службой HR, работает в компании 2 года, что в несколько раз меньше, чем средний стаж работы наших выпускников.

#### **4. Во что это обходится**

Для иллюстрации описанного выше подхода участия предприятия в подготовке студентов приведем конкретные затраты ЗАО «Ланит-Терком» в 2006-2007 учебном году.

Сотрудники предприятия вечерами ведут занятия со студентами отделения информатики мат-мех факультета СПбГУ. Иногда учатся и студенты других отделений мат-меха или других факультетов, изредка встречаются и студенты других вузов, мы никаких ограничений не вводим.

Каждый департамент выделяет сотрудников для работы со студентами в рамках своих интересов и технологий. Студенты имеют возможность попробовать себя в нескольких группах, пока не нащупают близкие им темы. Сначала в каждой группе проводится несколько занятий в традиционном для университета стиле (лекции, семинары), затем формируются студенческие проекты для отработки навыков работы в команде. Каждым проектом руководят не менее двух

сотрудников – для страховки (командировки, срочные работы, болезни). Обычно проект длится три месяца, темы носят скорее развлекательный, чем серьезный характер, но организация проекта и используемые технологии носят вполне промышленный характер (планирование, еженедельные отчеты, версионный контроль, контроль качества и т.п.) Темы работ согласовываются с кафедрой системного программирования, для студентов отделения информатики работа в студенческом проекте может быть зачтена в качестве курсовой работы (разумеется, при успешном завершении).

По завершении проекта устраивается публичная презентация, причем каждый студент должен рассказать о своем участии в проекте. Умение представить свои результаты, правильная речь и грамотность входят в обязательный перечень знаний по программной инженерии. Интересно, что практически в каждом проекте в первые же недели образуются один-два лидера, их никто не назначает, лидерские качества – это свойства характера. Лидеров мы берем на особую заметку, из них вырастают хорошие руководители.

Для лучших студентов из всех проектов мы организуем летние школы в июле (4 часа занятий в компьютерном классе в день). Поскольку эти студенты жертвуют частью своего отпуска, мы им платим стипендию (2000 рублей плюс 2500 рублей в случае успешной работы на школе).

В 2006-2007 учебном году было образовано 9 студенческих проектов, в них начали работать 80 студентов, к моменту завершения проектов осталось 60 студентов, что мы считаем большим успехом. В первые годы нашего эксперимента завершало работу менее половины студентов. Сейчас мы предлагаем более интересные проекты, да и общая организация за эти годы сильно выросла.

Каждый сотрудник, руководивший проектом, тратил на эту деятельность примерно 6 часов в неделю и получал за это дополнительно к зарплате 3-6 тысяч рублей в месяц. Потеря шести часов в неделю заметна для основной работы, но руководство департаментов сознательно идет на эти потери в расчете на приход хороших кадров.

Всего в описываемом учебном году на подготовку студентов предприятие затратило около \$30 тысяч, было принято на стажировку 30 человек, т.е. порядок затрат – \$1000 на одного стажера. Разумеется, стажер – это ещё не готовый специалист, его, так же как и нового специалиста, найденного службой HR, надо доучивать, на него будут тратить свое время

другие сотрудники департамента, но профессиональный уровень получающегося в конце концов специалиста очень высок. В ЗАО «Ланит-Терком» практически все руководители групп, отделов, департаментов – это наши выпускники.

Наша компания быстро растет, плановая подготовка специалистов из числа студентов не может закрыть все потребности в кадрах, кроме того, иногда нужны специалисты в таких областях, в которых мы подготовку не ведем. Наша служба HR ежегодно набирает несколько десятков человек, например, в том же 2006-2007 году было нанято 20 человек, при бюджете HR службы в \$50 тысяч. Таким образом, на практике мы используем оба метода и имеем возможность их сравнивать.

## **5. Как удерживать кадры**

Разумеется, для каждого наемного работника важна величина зарплаты. Но не только зарплата удерживает работника на предприятии. Например, в ЗАО «Ланит-Терком» мы несколько раз проводили анонимные опросы сотрудников с целью понять их приоритеты, почему они работают именно у нас. Получилась такая картина:

1. Большая стабильная компания с большой историей
2. Тесная связь с СПбГУ, наукоемкое производство
3. Большой набор вертикалей, экспертиз, дающий сотруднику широкий выбор по интересам
4. Высокое качество менеджмента, хорошо налаженные процессы
5. Конкурентно-способная зарплата
6. Социальный пакет
  - оплата компанией добровольного медицинского страхования сотрудников
  - бесплатное обучение иностранным языкам
  - «белая» зарплата, оплата отпусков, больничных листов
  - корпоративные праздники

Всё это позволило нам уменьшить текучесть кадров за последние 5 лет с 12% до 8%.

Мы верим, что сотрудники уходят с предприятия не из-за «более зелёной травы» где-то там, на других предприятиях, а из-за недостаточной зелёной травы (по меркам конкретного рынка труда) на данном предприятии. Для удержания кадров, по сумме перечисленных выше факторов, предприятие должно

оказаться конкурентноспособно в глазах сотрудника с другими аналогичными предприятиями города.

Как мы видим из опросов, индекс привлекательности предприятия составляется из материальной и нематериальной частей. Средние значения и ожидания от полного пакета компенсации (зарплаты плюс социальных благ) будут примерно одинаковыми для всех сотрудников одного уровня, откуда бы они ни пришли на предприятие. Значит, дело в нематериальных факторах.

Для сотрудников, пришедших «с улицы» фактор 2 (связь с СпбГУ и наукоемкость) может не иметь такой привлекательности, как для выращенных на кафедре кадров, а фактор 3 (менеджмент и процессы, принятые на предприятии) потребует адаптации по сравнению с предыдущим местом работы – а это всегда дискомфортно. Для тех, кто пришел на предприятие, будучи уже знакомыми с принятыми там стандартами, оба фактора позитивны и составляют уже впитанный ими «дух предприятия». Идентификация сотрудников с ярко выраженным направлением, методами и ценностями предприятия и составляют ту разницу в средней продолжительности работы сотрудника, о которой мы говорим.

## **6. Заключение**

В этой статье авторы рассмотрели экономические вопросы, связанные с набором кадров начального уровня на предприятия в области ИТК, в условиях нынешнего рынка квалифицированных ИТК специалистов в России.

Мы показали, что при следовании обычным методам найма «с улицы» добиться окупаемости кадров начального уровня за то время, что они проведут на первичной должности, сложно, и требует жесткого контроля бюджета по найму, с возможными негативными последствиями для качества пере- и дообучения нанятых кадров.

С другой стороны, путь сотрудничества в области подготовки студентов между вузом и предприятием ведет к успешному обеспечению притока молодых специалистов на предприятие с одновременным существенным (в 4-5 раз) сокращением затрат. Сотрудничество будет продуктивным и взаимовыгодным только при соблюдении определенных условий – как мы подробно рассмотрели в параграфе 3.

Мы проанализировали пример подобного сотрудничества между СПбГУ и предприятием «Ланит-Терком» и произвели расчеты сравнительных затрат предприятия. В заключение, мы коснулись вопроса удержания специалистов на предприятии и влияния «происхождения» кадров на их лояльность к предприятию (выраженную в сумме факторов, выбранных самими сотрудниками по опросам).

Мы надеемся, что эта статья заинтересует лидеров предприятий ИТК рынка и побудит рассчитать прибыльность и экономическую эффективность процессов найма сотрудников с учетом «перегретости» нынешнего рынка труда в России.



АЛЬТЕРНАТИВНЫЙ ВЗГЛЯД НА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ  
ПО КОНТРОЛЮ КАЧЕСТВА.

**100% готовность.**

Альтернативный взгляд это всегда «Пре-»  
Предосторожность. Предусмотрительность. Предупреждение. Призван  
стать настоящим потрясением для конкурентов.

Он проактивно обеспечивает высокое качество работы приложений  
для поддержки бизнес-процессов.

Это заблаговременное понимание и сочетание задач – поскольку подход  
«Ну, я полагаю, нам это нужно» уже не работает.

Это программное обеспечение по контролю качества создано для  
устранения неопределенности и рисков при развертывании  
и обновлении системы.

Технологии успеха в бизнесе. [www.hp.ru/managementsoftware/QM](http://www.hp.ru/managementsoftware/QM)





# Agile Requirements Management

Askhat Urazbaev

## Abstract

At first sight it looks like being Agile means get rid of requirements analysis. For many people using agile methodology is merely an excuse for lacking documentation and analysis in the project. Indeed, once we have a rapid feedback from customer, is it really important to waste time on thorough requirements elaboration? Instead we can build "something", then allow our customer to look at it and rework it if needed.

In the real world this approach often drives to serious problems. During the session we will discuss the need in analysis in Agile and when it is needed. We will consider different requirement elaboration practices and discuss requirements management process in agile world.

## Content

- Do we need requirements analysis in Agile?
- Agile requirements practices and artifacts
- Requirements elaboration and analysis lifecycle
- Metrics



**Bio:** Askhat Urazbaev has graduated from MIPT and made a typical career from junior developer to project manager. Later on, in Luxoft, Askhat as a process architect was engaged in adopting "heavyweight" methodologies in different departments of the company. Happily, thanks to one of the customers, Askhat got familiar with Agile.

The new way of development looked very effective. Askhat has participated in several Agile projects. Askhat started helping teams and departments to adopt Agile methodologies in Luxoft. In March 2006 he has started Russian Agile Community (AgileRussia.ru). In 2008 Askhat along with his partner Nikita Filippov founded company ScrumTrek which consults, trains and helps companies in their transition to Agile. Among the customers of ScrumTrek there are such companies as Yandex, Afisha, HeadHunter, Ascon, Luxoft, Infopulse and many others.

# Управление требованиями в Agile

Асхат Уразбаев

## Тезисы

На первый взгляд кажется, что Agile означает отсутствие анализа требований. Для многих Agile - способ оправдать отсутствие документации и анализа в проекте. Но действительно, если мы можем получать быструю обратную связь от заказчика, так уж нужно ли заниматься сбором и анализом требований? Сделаем "что-то", затем покажем результат заказчику и, если потребуется, перделаем.

На практике такой подход часто приводит к серьезным проблемам. В докладе мы порассуждаем на тему того, нужен ли анализ в Agile и если нужен - то в каких случаях. Мы посмотрим на различные практики анализа требований и обсудим процесс работы с требованиями в Agile.

## Содержание

- Нужен ли анализ требований в Agile
- Практики и артефакты анализа
- Жизненный цикл сбора и анализа требований
- Метрики

**Биография:** Асхат Уразбаев закончил в 2000 году с отличием Московский Физико-Технический Институт и сделал вполне типичную карьеру от младшего разработчика до менеджера проектов в небольшой аутсорсинговой компании. Позже, в компании Luxoft, Асхат в качестве архитектора процессов занимался внедрением «тяжелых» традиционных методологий разработки программного обеспечения в подразделениях компании. Благодаря одному из заказчиков он «познакомился» с Agile. Новый способ разработки оказался чрезвычайно эффективным. Затем были еще и еще проекты, осуществленные с использованием принципов Agile. Асхат стал заниматься внедрением гибких методологий в подразделениях компании Luxoft. В марте 2006 года он основал движение практиков гибкой разработки AgileRussia. В 2008 году Асхат вместе с партнером Никитой Филипповым основал компанию ScrumTrek, которая консультирует, проводит тренинги и помогает компаниям внедрять гибкие методологии. Среди

клиентов ScrumTrek такие компании, как Yandex, HeadHunter, Afisha, Acronis, Luxoft, Infopulse и многие другие.

# VMware Infrastructure – the New Distributed Operating System for the Datacenter

**Bogomil Balkansky**  
**VMware**

## **Abstract**

Tens of thousands of companies around the world are standardizing on VMware's flagship suite VMware Infrastructure to dramatically decrease the cost of computing, align IT infrastructure with business requirements, and provide unprecedented risk mitigation and resilience. This session will discuss VMware's roadmap of groundbreaking new products and technologies that expand the VMware Infrastructure suite into a virtual datacenter operating system (VDC-OS). The virtual datacenter OS from VMware allows businesses to efficiently pool all types of hardware resources - servers, storage and network – into an aggregated on-premise cloud to deliver a highly elastic, self-managing and self-healing datacenter. The session will also discuss how the virtual datacenter OS enables federation of on-premise and off-premise infrastructure to enable cloud computing.



**Bio:** Bogomil leads the product marketing team for VMware Infrastructure – VMware's flagship suite for the datacenter. Bogomil brings more than 10 years of experience defining and marketing application and infrastructure software solutions.

Prior to joining VMware, Bogomil was Group Manager in the Customer Data Integration business unit at Siebel Systems, where he played a key role in growing the business from inception to more than \$75 million in revenue over three years, capturing leading market share. Prior to Siebel Systems, Bogomil was a product manager at CrossWeave, where he worked on defining a composite applications platform. Bogomil began his career as a management consultant at McKinsey & Company, where he advised financial services companies on strategic and operational issues.

Bogomil holds a Bachelor of Arts in Mathematics from Cornell University and an MBA from Stanford Graduate School of Business.

# **VMware Infrastructure – Новая распределенная операционная система для центров обработки данных**

**Богомил Балканский  
VMware**

## **Тезисы**

Десятки тысяч компаний во всем мире смогли снизить расходы на поддержание ИТ-инфраструктуры и согласовать ИТ-инфраструктуру с бизнес-требованиями, используя VMware Infrastructure - флагманский продукт VMware. В этом докладе будет обсуждаться план развития VMware Infrastructure, который позволит расширить этот программный пакет до полноценной виртуальной операционной системы для центров обработки данных (VDC-OS). Программный пакет VMware Infrastructure предназначен для виртуализации серверов, хранилищ и сетей, а также для преобразования ИТ-инфраструктуры в автоматизированную и стабильную вычислительную среду. Это наиболее обширный программный пакет для виртуальных инфраструктур, с помощью которого вы можете немедленно приступить к созданию само-оптимизирующейся ИТ-инфраструктуры.

**Биография:** Богомил Балканский руководит департаментом маркетинга VMware Infrastructure - флагманского продукта VMware для центров обработки данных. До прихода в VMware, Богомил был руководителем группы в отделении Customer Data Integration в компании Siebel Systems, где он сыграл ключевую роль в развитии бизнеса от его начала и до более чем \$ 75 млн. дохода в течение трех лет, и вывода бизнеса на лидирующие позиции на рынке. До работы в Siebel Systems, Богомил был менеджером по продукции в CrossWeave, где он работал над разработкой платформ композитных приложений. Богомил начинал свою карьеру как консультант по вопросам управления в компании McKinsey, где он предоставлял консультационные услуги по стратегическим и операционным вопросам. Богомил получил степень бакалавра в области математики в Корнеллском университете и MBA в Стэнфордской Высшей школе бизнеса.

# **Globalization of software development in matrix management environment.**

**Igor Agamirzian**  
**EMC Corporation**

## **Abstract**

The paper presents management model of distributed software development in the current environment, efficiency and challenges of proposed approaches are compared. Different models ranking between fully centralized and completely decentralized ones are compared. Optimization of process, balancing approaches based on existing skills and domains of expertise is proposed. Experience of implementation of distributed software development model within MNC based on of EMC practice is considered.



**Bio:** Igor Agamirzian, Ph.D., is a General Manager of the Software Development Center of EMC Corporation in St. Petersburg, Russia. In his previous position he was a National Technology Officer and a member of the Cabinet of the Chairman of Microsoft in Russia and CIS countries. He was running Government Relations and

Education programs for Microsoft in 2002-2004, prior to that he was responsible for managing relations for Microsoft Research with Computer Science research institutions in the Eastern Europe, including Russia and other CIS countries during 1999-2002.

Before joining European Microsoft Research Lab in Cambridge, UK, Igor Agamirzian was running enterprise sales of the Microsoft Moscow subsidiary. He joined Microsoft in 1995 as a consultant of the Microsoft Consulting Services, and in 1996 started MCS practice in Moscow, Russia. Before joining business in 1991 as a co-founder and Technical Director of Astro Soft Ltd., a St. Petersburg based software development and system integration company, Igor Agamirzian had a 12-year background as a senior research fellow in the field of programming languages and computer architecture with the leading institutes of the Academy of Science of the USSR.

Igor Agamirzian is an active Russian IT promoter within the international research and business community. In 1991 he published a first overview of the history and current state of the Soviet Computer Science in the widely known computer magazine “Computing in the U.S.S.R.” (Byte, April 1991). He is strongly involved in the international activities around Information and Communication Technologies, representing Russian private sector in the G8 DOT Force, an international taskforce on the Digital Divide and Information Society, and is often speaking on the business and research conferences. In year 2002 Igor Agamirzian was appointed as a Member of Advisory Board of the United Nations ICT Task Force.

# Глобализация разработки программного обеспечения в условиях матричного управления

Игорь Агамирзян  
ЕМС Corporation

## Тезисы

В докладе рассматривается управленческая модель распределенной разработки программного обеспечения в современных условиях, оцениваются эффективность и вызовы предлагаемых подходов. Сравниваются различные варианты управления от полностью централизованной до сетевой модели. Предлагается оптимизация процесса, базирующаяся на нахождении баланса на основе имеющихся навыков и экспертизы. Практика реализации модели распределенной разработки программного обеспечения в крупной транснациональной корпорации рассматривается на примере опыта корпорации ЕМС.

**Биография:** Игорь Рубенович Агамирзян, к. ф.-м. н., является Генеральным директором Центра разработки программного обеспечения корпорации ЕМС в Санкт-Петербурге, Россия. На своем предыдущем месте работы он занимал должность Директора по стратегии в области науки и технологий и члена Кабинета Президента Microsoft в России и СНГ. В 2002-2004 годах он руководил программами Microsoft по образованию и работе с правительственными организациями, до этого (1999-2002) отвечал за взаимодействие с университетами и научно-исследовательскими организациями в Восточной Европе, являясь сотрудником Microsoft Research.

До работы в лаборатории Microsoft Research в Кембридже, Великобритания, Игорь Рубенович Агамирзян руководил отделом по работе с корпоративными заказчиками в Представительстве Microsoft в Москве. Он начал работу в Microsoft в качестве консультанта Microsoft Consulting Services в 1995 году, и в 1996 году организовал практику MCS в Москве. До начала работы в бизнесе в 1991 году в качестве соучредителя и технического директора компании АстроСофт в Санкт-Петербурге, Игорь Рубенович Агамирзян в течение 12 лет работал в качестве научного сотрудника в ведущих институтах Академии Наук, занимаясь исследованиями в



области языков программирования и архитектуры компьютеров.

Игорь Рубенович Агамирзян известен как активный пропагандист российских ИТ за рубежом, как в бизнесе, так и в науке. В 1991 году он опубликовал первый обзор истории и современного состояния программирования в СССР в массовом компьютерном издании – статья “Computing in the U.S.S.R.” (журнал Byte, апрель 1991). Он активно участвует в международной деятельности в области информационных и коммуникационных технологий, часто выступает на научных и бизнес-конференциях. Игорь Рубенович Агамирзян был российским представителем в G8 DOT Force, международном экспертном совете по проблемам информационного общества и преодоления информационного неравенства. В 2002 году он был назначен советником ООН по ИКТ в UN ICT Task Force.



**EMC<sup>2</sup>**  
where information lives®

Корпорация EMC – ведущий мировой разработчик и поставщик технологий и решений для информационных инфраструктур. Решения корпорации позволяют организациям любых размеров кардинально усовершенствовать информационную инфраструктуру, приобрести конкурентные преимущества и, в конечном итоге, улучшить свое положение на рынке.



Санкт-Петербургский Центр Разработок корпорации EMC является частью глобальной программы EMC по инвестициям в исследования и разработки. Центр занимается созданием и развитием широкого спектра лидирующих на рынке программных продуктов и ключевых технологий компании, таких как:

- **платформы систем хранения данных**
- **решения для архивирования больших объемов данных и устранения избыточности данных**
- **системы ввода информации и электронного документооборота**
- **решения по интеграции систем хранения с платформой Mainframe**

За подробной информацией обращайтесь на сайты [www.russia.emc.com](http://www.russia.emc.com) и [www.emc-spb.com](http://www.emc-spb.com)

Информация о вакансиях доступна на странице [www.russia.emc.com/go/jobs](http://www.russia.emc.com/go/jobs)

и по электронной почте [jobspb@emc.com](mailto:jobspb@emc.com).

ООО «Санкт-Петербургский Центр Разработок EMC» – Российская Федерация

199004, Санкт-Петербург, ВО, Средний пр., 36/40, Бизнес-центр «Остров»

тел. +7 (812) 325-4633, факс +7 (812) 325-4607

# The Future of Multi-core: Intel's Tera-Scale Computing Research

**Jim Held**  
**Intel**

## **Abstract**

The Intel® Tera-scale Computing Research Program is a worldwide research effort to create platforms for the next decade with capabilities only dreamed of today. This requires embracing a shift to massive parallelism through scalable multi-core architectures, platforms and software which use 10s to 100s of cores to efficiently process hundreds of threads and terabytes of data. This presentation is about Intel's research vision and the hardware and software programming research to advance it.



**Bio:** Jim Held is an Intel Fellow who leads a virtual team of architects conducting Tera-Scale Computing Research in Intel's Corporate Technology Group. Since joining Intel in 1990, he has led research and development in a variety of Intel's architecture labs concerned with media and interconnect technology, systems software, multi-core processor architecture and virtualization. Before coming to Intel, Jim worked in research and teaching capacities in the Medical School and Department of Computer Science at the University of Minnesota where he earned a Ph.D. (1988) in Computer and Information Science.

# Будущее много-ядерности: программа исследований Tera-Scale Computing

Джим Хелд  
Intel

## Тезисы

Программа исследований Tera-Scale Computing компании Intel это всемирный проект корпорации Intel, направленный на разработку компьютерных платформ будущего, о возможностях которых сегодня можно только мечтать. Это потребует перехода к параллелизму с помощью масштабируемых многоядерных архитектур, платформ и компьютерных систем, основанных на процессорах с десятками и даже сотнями вычислительных ядер, для того, чтобы эффективно обрабатывать сотни потоков и терабайты данных. Эта презентация о направлениях исследований компании Intel и программном обеспечении для их усовершенствования.

**Биография:** Джим Хелд работает в подразделении Corporate Technology Group компании Intel и имеет статус Intel Fellow. В настоящее время он возглавляет виртуальную команду архитекторов по программе исследований Tera-Scale Computing. С тех пор как он пришел в Intel в 1990 году, он руководил исследованиями и разработкой ряда проектов Intel, занимающихся вопросами медиа и связующими технологиями, системного программного обеспечения, архитектуры многоядерных процессоров и виртуализации. До прихода в Intel, Джим занимался научно-исследовательской и преподавательской деятельностью на медицинском факультете и факультете компьютерных наук в университете штата Миннесота, где он получил Ph.D. степень (1988) в области компьютерных и информационных наук.

# Open Source in Russia: Risks and Opportunities

**Viktor Ivannikov**

**Institute of Systems Programming of Russian Academy of Sciences**

**Bio:** Viktor Ivannikov is an academician of Russian Academy of Sciences, director of the Institute of Systems Programming, Head of the Department of Systems Programming at the Moscow State



University and Moscow Institute for Physics and Technology, chief editor of the «Programming» magazine. Viktor is a member of the international scientific community ACM, IEEE Computer Society; also he heads the Russian Branch of IEEE Computer Society.

Viktor Ivannikov has been a director of the Institute of Systems Programming (Russian Academy of Sciences) since its founding in January 1994. Ivannikov has published more than 100 scientific papers in the field of development of the architecture of computers and computer systems, interoperability of distributed object-oriented systems. Victor made a fundamental contribution to development of operating systems for processing information in real time for the space flight control centers.

# **Свободное программное обеспечение: Проблемы и перспективы**

**Виктор Иванников**  
**Институт системного программирования РАН**  
**ivan@ispras.ru**  
**www.ispras.ru**

**Биография:** Виктор Петрович Иванников — академик РАН, директор Института системного программирования РАН, заведующий кафедрами системного программирования на факультете вычислительной математики и кибернетики МГУ и в МФТИ, главный редактор журнала «Программирование». Виктор Петрович является членом международных научных сообществ ACM, IEEE Computer Society; возглавляет Российское отделение IEEE Computer Society.

В. П. Иванников является директором Института системного программирования РАН со времени образования ИСП в январе 1994. Являлся одним из основных участников создания первой операционной системы (Д-68) для ЭВМ БЭСМ-6. Опубликовал более 100 научных работ в области разработки архитектуры ЭВМ и вычислительных комплексов, обеспечения интероперабельности в распределенных объектно-ориентированных системах. Внес фундаментальный вклад в создание теории и практики разработки операционных систем вычислительных комплексов, обеспечивающих обработку информации в режиме реального времени в центрах управления полетами космических аппаратов.

## **Тезисы**

### **Классификация СПО проектов**

Свободное программное обеспечение (СПО) — это не только исходные тексты программ или программы в двоичном виде, в виде готовом для исполнения на компьютере. СПО — это еще и процессы разработки и сопровождения программ; это сообщества создателей и пользователей СПО; это новые модели бизнеса и многие другие стороны развития экономики, культуры, юриспруденции и общества в целом. По-видимому, нет большого смысла классифицировать по видам собственно

свободных и открытых программ, так как этих видов много. Зато можно утверждать, что СПО проекты, как правило, можно отнести к одному из четырех видов. Это:

- индивидуальные,
- университетские,
- корпоративные,
- общественные

Первые создаются одним энтузиастом; иногда к нему присоединяется еще небольшое число добровольцев. Время активной жизни проекта не велико и число пользователей также не велико.

Второй вид – это проекты, которые ведутся университетскими командами. В начале проекта ядро команды ограничивается сотрудниками лаборатории; в дальнейшем к команде может присоединиться значительное число энтузиастов. Университетские команды существенно более стабильные и долгоживущие, нежели стихийные группы энтузиастов, больше и количество пользователей. Часто складывается альянс из нескольких университетов, что в дальнейшем упрощает построение сообщества, члены которого объединены целями и работами проекта.

Корпоративные проекты СПО появились не так давно и поначалу воспринимались как нонсенс, так как считалось, что СПО – это поле, открытое в первую очередь для энтузиастов и альтруистов, а совсем не для бизнеса и серьезных корпораций. Сейчас этот стереотип многими уже не признается, имеется целый ряд примеров успешных корпоративных СПО проектов. Главное отличие корпоративных проектов состоит в том, что проект преследует вполне определенные цели поддерживающей его корпорации (или консорциума корпораций), проект как правило имеет хорошо организованный менеджмент и стабильное финансирование. Примерами успешных корпоративных СПО проектов являются Eclipse (IBM) и Mozilla (AOL).

Последний, четвертый вид проектов – проекты, вокруг которых образовались сообщества разработчиков и пользователей; соответственно, основной движущей силой этих проектов являются коллективные усилия разработчиков в реализации целей, которые также определяются сообществом. «Общественные» проекты – это высшая форма СПО проектов. Они имеют наибольшее число пользователей, их развитие стабильно, несмотря на очевидные сложности вычленения общих целей из множества разнонаправленных целей членов

сообщества и проблемы с интеграцией вкладов отдельных участников в общее дело. Примерами успешных «общественных» СПО проектов являются GCC, GNU Linux, Apache.

### **Мифы о СПО**

СПО, как явлению многостороннему и постоянно развивающемуся и изменяющемуся, трудно, даже невозможно, дать полное определение. Мы также не будем этого делать, однако остановимся на характерных особенностях СПО, которые часто порождают противоречивые толкования. Их часто называют «мифами». Остановимся на трех из них:

- Никто не управляет развитием СПО и каждый может внести свои правки – и то и другое ведет к нестабильности и небезопасности СПО; у СПО нет служб поддержки;
- СПО против коммерциализации;
- СПО используется только хакерами и технической элитой.

В мире существует масса неуправляемых проектов, которые создают (если вообще создают) нестабильные программные продукты, которые, в свою очередь, никто не поддерживает. Но это, в первую очередь, проекты первых двух видов, то есть проекты, которые не переросли в стадию корпоративных или «общественных». В отношении двух последних нужно сказать, что уровень как собственно продуктов, так и процессов разработки и сопровождения в них достаточно высок; в частности, он устраивает вполне серьезных и разборчивых корпоративных пользователей.

На ранних стадиях развития СПО модели, считалось, что сам по себе программный продукт должен быть бесплатным, зато услуги по его адаптации, оптимизации, сопровождению могут быть платными, и это должно является источников ресурсов для обеспечения жизнедеятельности СПО проектов. Сейчас такая форма «коммерциализации» СПО не отвергается, но для крупных проектов, как корпоративных, так и «общественных», существенную долю ресурсов предоставляют крупные корпорации и государственные структуры. Это объясняется тем, что успешное развитие таких проектов отвечает стратегическим интересам и тех и других, ровно по этой причине они вкладывают деньги в такие проекты.

СПО доступно для хакеров – поскольку здесь все открыто и бесплатно. Техническая элита также может извлекать уникальные преимущества СПО. Интересный пример – Microsoft поддерживает изучение Linux в университетах и преимущественно набирает программистов для развития своей



ОС из тех, кто имеет хороший опыт в Linux. Однако сейчас, когда тот же Linux получил хорошее оснащение для desktop приложений, он начал использоваться в офисах, где весь персонал не является даже профессиональными программистами.

### **О государственной политике поддержки СПО**

Одним из уникальных преимуществ СПО является то, что СПО благотворно влияет на многие процессы жизни общества, и, в частности, на экономику. СПО «как бы автоматически» создает условия соревновательности в промышленности информационных технологий и является средством противодействия монополизации. Поскольку развитие соревновательности в экономике, очевидно, отвечает интересам развития современного общества и нашей страны, в частности, наше государство, как и многие другие (в первую очередь европейские, Корея и Китай) развертывают программы поддержки СПО. Такая поддержка крайне важна для развития СПО и ИТ индустрии в целом. Крупные СПО проекты становятся успешными только при наличии внешней поддержки, и, если нет поддержки корпораций, роль главного спонсора должно брать на себя государство.

Вместе с тем, государственная поддержка приносит огромные риски. В России эти риски особенно серьезны. При массовом развертывании ИТ проектов, которые полностью дотируются государством, легко ожидать, что будет сделано много низкокачественного ПО. Такая опасность имеется при использовании любой модели разработки, как СПО, так и проприетарной. Однако в СПО заложен механизм объективной оценки качества работы программистов, который особенно хорошо работает в «общественных» проектах. Критерием качества является принятие сообществом разработки, выполненной его членами. В крупных проектах сообщества, как правило, интернациональные, соответственно, оценки сообщества вполне объективны.

Еще одно важное преимущество, обусловленное интернациональным характером СПО сообществ, состоит в том, что в СПО проекте участники, живущие в разных странах, и зачастую работающие на различные компании и организации, хорошо узнают друг друга. Каждый активный участник СПО проекта работает не только на проект, он работает на свою репутацию. Хорошая репутация (как в части технических знаний и умений, так и в организационном и коммуникативном

плане) – это эффективный инструмент для построения бизнес-контактов, организации новых совместных, в том числе и коммерческих проектов. Тем самым, поддержка СПО государством непосредственно работает на развитие высокотехнологичной сферы экономики и, в частности, на развитие экспортного потенциала страны.

### **Заключение**

Уже сейчас мы являемся свидетелями того, что СПО выходит из фазы экспериментальных проектов, базирующихся преимущественно на энтузиазме участников, и переходит в фазу зрелых проектов, создающих добротные программные продукты и опирающиеся на зрелые процессы их разработки.

Сейчас СПО нельзя рассматривать только как еще один вид ПО или даже как еще одну модель разработки ПО. СПО это новый феномен, влияние которого распространяется на многие стороны экономики, юриспруденции, образования и другие составляющие жизни общества. Тем не менее, центральное место в списке возможностей, которое предоставляет СПО обществу, является триада «технологии-бизнес-образование». СПО, благодаря своей открытости и доступности, предоставляет уникальные возможности для развития и внедрения в жизнь новых технологий. СПО обеспечивает конкурентную, соревновательную среду, поддерживающую развитие бизнеса. СПО – уникальная возможность по обучению профессионалов в информационных технологиях, владеющих самыми передовыми методами и инструментами. При этом все три перечисленные возможности несколько не противоречат друг другу, а, напротив, демонстрируют синергетический эффект, взаимно усиливая друг друга.

Перечисленные соображения позволяют сказать, что сейчас складывается новая система взаимоотношений между СПО, с одной стороны, и государством, бизнесом и другими общественными институтами, с другой. Государство и бизнес не только становятся потребителями результатов СПО-продуктов. Эти общественные институты начинают рассматривать СПО как инструмент для решения своих стратегических задач. Это еще более укрепляет позиции СПО и открывает новые перспективы перед обществом.

# **Yesterday - File Managers. Today - Web Browsers. What will We Crown Tomorrow?**

**Vladimir L. Pavlov**  
**International Software and Productivity Engineering Institute**

## **Abstract**

15 years ago almost everyone was using file-managers (such as Norton Commander). The time has changed - today we spend the most of our time with web-browsers. This will not last forever - get prepared to the upcoming global climate change in the software world...



**Bio:** Vladimir L. Pavlov is the chairman and chief strategy officer of the International Software and Productivity Engineering Institute (INTSPEI). He founded INTSPEI ([www.intspei.com](http://www.intspei.com)) to launch new software development methodologies resulting from his experiments and research.

A leading expert in software development, Vladimir has previously served as director and/or CTO for top high-tech companies, including Intel and Microsoft, in the US, Ukraine, Russia, and Poland.

A frequent speaker at scientific and industrial conferences, he has authored major publications on computer science and software engineering. For more information about the International Software and Productivity Engineering Institute, visit [www.intspei.com](http://www.intspei.com) .

For more information about Vladimir L. Pavlov, visit [www.vlpavlov.com](http://www.vlpavlov.com)

# **Вчера - файловые менеджеры. Сегодня - веб-броузеры. Что ждет нас завтра?**

**Владимир Л. Павлов**  
**Международный НИИ Проблем программирования**  
**INTSPEI**

## **Тезисы**

Всего 15 лет назад почти каждый пользователь компьютера постоянно работал с каким-то из файловых менеджеров (например, Norton Commander). Сегодня времена поменялись, и софтверным миром стали править веб-броузеры. Надолго ли?

**Биография:** Владимир Л. Павлов - международно-признанный эксперт в области управления разработкой ПО, имеющий за плечами почти два десятка лет опыта в создании и эксплуатации комплексных программных решений. В данный момент Владимир является директором Международного НИИ проблем программирования INTSPEI ([www.intspei.com](http://www.intspei.com)).

Работая техническим директором в ряде крупных софтверных компаний (включая Intel, Microsoft, и другие), Владимир успешно инициировал и руководил реализацией инновационных R&D проектов, выполнявшихся по заказу крупных международных корпораций высокотехнологического сектора в Польше, России, Украине и США.

Владимир Л. Павлов является автором/соавтором большого количества публикаций в области информатики и программной инженерии и часто приглашается с выступлениями по данной тематике на научные и индустриальные конференции.

# SDLC Fine-Tuning

## Optimize Your Software Development Life Cycle!

This specialized consulting service is intended to help software companies:

- Improve their efficiency;
- Enable engineers to eliminate bugs early;
- Shorten overall development time;
- Improve software quality.



## Our SDLC Fine-Tuning Service Includes Three Major Phases:

- Identify bottlenecks in the software development process currently being used;
- Design and implement process improvements addressing identified issues;
- Teach software engineers how to effectively implement and use these process changes.

Please write us at [info@intspei.com](mailto:info@intspei.com) to request the service or to learn more about INTSPEI solutions.



International Software and Productivity Engineering Institute  
1979 Marcus Avenue, Suite 210, Lake Success, NY 11042, USA  
Call: +1-877-INTSPEI (+1-877-468-7734)  
Visit: [www.intspei.com](http://www.intspei.com)



# Parallel Programming 2.0

**Wei Li**  
**Intel**

## **Abstract**

Multi-core has replaced GHz as the driver of performance and it presents software developers with a new challenge: programming for concurrency. Parallel programming in the mainstream has different characteristics than in the previous era when it was restricted to high performance computing. This presentation will cover this new paradigm, its challenges, and software products being developed by Intel.



**Bio:** Wei Li is an Intel Sr. Principal Engineer and Director of Emerging Products Lab in Intel's Software and Solutions Group. He is responsible for developing software products in the areas of threading tool, binary translation, and compiler. He manages software teams across US, Russia and China. In his 10 years at Intel, he won 2 Intel Achievement

Awards for developing innovative compiler technologies and achieving performance leadership on Intel Architectures. He worked extensively with ISV's, OEM's, and major Intel customers to enable the sales of Intel hardware platforms and software tools. He received a Ph.D. in computer science from Cornell University, and taught graduate compiler courses at Stanford University and the University of Rochester. He is an Associate Editor of ACM Transactions on Programming Languages and Systems (TOPLAS).

## Параллельное программирование 2.0

Вэй Ли  
Intel

### Тезисы

Многоядерность заменила гига-герцы как двигатель производительности и теперь ставит новую задачу разработчикам: программирование для параллельных процессов. Параллельное программирование в настоящем понимании имеет характеристики отличные от тех, которые оно имело в прошлой эре, ограниченное высокопроизводительными вычислениями. Эта презентация расскажет о новой парадигме многоядерных вычислений, ее проблемах и о продуктах, разрабатываемых Intel для развития многопоточного ПО.

**Биография:** Вэй Ли является ведущим инженером подразделения Software and Solutions Group компании Intel и директором Emerging Products Lab. Он отвечает за разработку программных продуктов в таких направлениях как: реализация многопоточности, бинарные трансляторы и компиляторы. Под его руководством работают команды разработчиков в США, России, Китае. За 10 лет работы в Intel, Вэй Ли был награжден двумя премиями: за разработку инновационных компиляционных технологий и за достижение лидерства архитектуры компании Intel в области производительности. Он много работал с независимыми поставщиками ПО, производителями комплексного оборудования и крупными клиентами Intel, чтобы запустить продажи аппаратных средств и программного обеспечения Intel. Вэй Ли получил степень Ph.D. в области компьютерных наук в Корнеллском университете, и преподавал курсы по компиляторам в Стэнфордском и Рочестерском университетах. Также он является помощником редактора журнала ACM Transactions on Programming Languages and Systems (TOPLAS).

# Adobe Rich Internet Application Platform

**Enrique Duvos**  
**Adobe Systems**  
**Senior Platform Evangelist**  
**email: [duvos@adobe.com](mailto:duvos@adobe.com)**

## Abstract

Adobe® Flex® Builder™ 3 Standard software is an Eclipse™ based development tool enabling intelligent coding; interactive step-through debugging; and visual design of user interface layout, appearance, and behavior of RIAs. The ability to import assets from Adobe Creative Suite® 3 makes it easy for designers and developers to work together.

- Adobe Flex applications can provide a highly interactive experience, seamlessly supporting the use of integrated media.
- Flex applications run identically in all major browsers and operating systems and can now run on the desktop with Adobe AIR™.

Flex provides a structured, proven, open-source development framework for the delivery of predictable, long-term RIA solutions, making Flex the preferred choice for web sites as well as intranet and enterprise applications.



**Bio:** Enrique Duvos is a Senior Platform Evangelist for Adobe in Europe. His role is to help customers, partners and developers across Europe understand and adopt the benefits and characteristics of some of the Adobe core platform technologies, such as Adobe Flash, PDF, AIR, Flex and LiveCycle ES. Prior to the Adobe merger

he was part of the Macromedia technical sales team specialized in the company's enterprise portfolio of technologies.

With a major in Computer Science and a Master's in distributed applications, his background includes more than ten years in the Software industry, five of them in R&D as Senior Software Architect designing and developing J2EE enterprise solutions for Macromedia US.



# Платформа Adobe Rich Internet Application

Энрике Дувос

## Тезисы

Adobe Flex Builder 3 Standard - это высокоэффективная среда разработки на базе Eclipse, которая позволяет создавать, отлаживать и поддерживать насыщенные веб-приложения. Возможность переносить данные из Adobe Creative Suite 3 позволяет дизайнерам и разработчикам легко и эффективно работать вместе.

- Приложения Adobe Flex могут предоставить богатый интерактивный опыт, поддерживающий использование встроенных данных.
- Приложения Flex совместимы со всеми наиболее распространенными обозревателями, платформами персональных компьютеров и версиями операционных систем, что достигается с помощью Adobe AIR.

Flex предоставляет структурированную среду разработки с открытым кодом для создания предсказуемых и долговременных насыщенных веб-приложений, делая Flex наиболее предпочтительным инструментом для создания веб-сайтов и корпоративных решений.

**Биография:** Энрике Дувос является ведущим специалистом компании Adobe в Европе. Его задача помогать заказчикам, партнерам и разработчикам по всей Европе понимать и использовать некоторых основных платформ Adobe, таких как Adobe Flash, PDF, AIR, Flex и LiveCycle ES. Перед работой в Adobe Энрике работал в компании Macromedia, в отделе технических продаж, специализирующемся на корпоративных решениях.

Кроме огромного опыта в компьютерных науках и степени магистра в области распределенных приложений, Энрике имеет более десяти лет практического опыта в программной индустрии. Около пяти лет Энрике посвятил работе в компании Macromedia US, где в качестве ведущего архитектора разрабатывал корпоративные решения на базе J2EE.

# Microsoft Security Development Lifecycle (SDL): What, Why and How

Ivan Medvedev  
Microsoft

## Abstract

The Microsoft Security Development Lifecycle is the industry leading software security assurance process. Since 2004 the SDL has been a mandatory policy within Microsoft and has proven to be an effective tool in embedding security and privacy into Microsoft software and culture. Given by a member of the team that helped make the SDL a reality, the talk will cover "the what", "the why" and "the how" of the Security Development Lifecycle.



**Bio:** Ivan Medvedev graduated from the Moscow State University and since 1999 has been working on various security technologies at Microsoft. Ivan has been a part of the group that is home to the SDL, MSRC and Secure Windows Initiative for four and a half years and currently leads a team of developers building internal security tools that help support the SDL process within the company.

# **Microsoft Security Development Lifecycle (SDL): что, как и почему**

**Иван Медведев  
Microsoft**

## **Тезисы**

Microsoft Security Development Lifecycle (SDL) является индустриальным лидером среди всех средств обеспечения безопасности процессов создания ПО. С 2004 года SDL является обязательным для использования внутри Microsoft, и зарекомендовал себя как эффективное средство органичной интеграции механизмов обеспечения безопасности в повседневные производственные процессы. Данный доклад представит “взгляд изнутри” команды, которая непосредственно создавала SDL, и ответит на основные вопросы, которые обычно задают об этом продукте.

**Биография:** Иван Медведев окончил МГУ в 1999, и с тех пор работает Рэдмонде над различными продуктами Microsoft в области безопасности. Иван работал над такими проектами, как SDL, MSRC и Secure Windows Initiative, и в данный момент возглавляет группу разработчиков, ответственную за создание инструментов обеспечения безопасности внутрикорпоративного процесса подготовки и выпуска новых продуктов Microsoft.

# Mediawiki: Silver bullet or Suisse knife?

**Stas Fomin**  
**Customized InformSystems Ltd.**

## Abstract

General-purpose Wiki systems are successfully used in last years for documenting and knowledge management in software companies. Though initially these systems were designed to aggregate knowledge in open communities, now Wiki systems often replace "stereotype" document management systems and software documenting technologies.

We will discuss some reasons for this situation and point out some problems that limit application scope of Wikis. Wiki is not a "silver bullet" but we can outline application domains where Wiki may be effectively used.

Then, we will consider how to choose Wiki system in order to build an internal corporate knowledge base, keep and maintain documentation and communicate with customers. MediaWiki is an optimal choice for this purpose in our opinion, since it has a vast community of users and developers and a wealth of experience. Open architecture makes it possible to extend easily any functionality of MediaWiki and to "downsize" possible inconveniences.

Finally, we will offer solutions for the most acute problems of "corporate wikiing".



**Bio:** Stas Fomin is a deputy of Director of Information Technology department in Custom InformSystems company. He joined this company in 1999 accomplishing the dozens of successful projects on the development and implementation of information systems. Stas is exploring the promising systems, frameworks, and standards, implementing effective tools and methodology of software development in his company.

Also Stas Fomin teaches algorithms and complexity theory at Moscow Institute of Physics and Technology (MIPT) and at the Institute for System Programming (ISP RAS). Stas is a co-author of

the book «Efficient algorithms and computational complexity», the author of big amount of scientific and engineering articles.

Prior joining Custom InformSystems Stas Fomin works as a developer of information systems for Server company. Stas Fomin graduated the Applied Mathematics faculty at Moscow Institute of Physics and Technology (MIPT) with honors in 1999.

# Mediawiki: Серебрянная пуля или швейцарский нож?

Стас Фомин  
Customized InformSystems Ltd.

## Тезисы

Сейчас вики-системы активно используются не только открытыми сообществами, но и компаниями-разработчиками для написания документации и внутрикорпоративного управления знаниями. При этом вики-системы вытесняют «классические» системы документооборота и технологии документирования. Мы рассмотрим причины этого и обозначим границы применимости, указав также на минусы вик, из-за которых они так и не стали «серебрянной пулей».

Затем мы рассмотрим выбор внутрикорпоративной вики-системы, где мы рекомендуем MediaWiki, имеющую огромный опыт использования и доработок, открытая архитектура которой позволяет, как легко расширять возможности, так и «срезать» неудобные углы в использовании.

В конце мы предложим решения наиболее острых проблем в использовании вики-систем.

**Биография:** Стас Фомин занимает должность замдиректора по информационным технологиям в компании «Заказные ИнформСистемы». В этой компании он работает с 1999 года, выполнив десятки успешных проектов по разработке и внедрению информационных систем. В последние годы Стас исследует перспективные системы, архитектуры, фреймворки и стандарты, внедряя в компании эффективные инструменты и методологии разработки ПО.

Также Стас Фомин преподает студентам курсы по алгоритмам и теории сложности в Московском физико-техническом институте (МФТИ) и Институте Системного Программирования (ИСП РАН). Стас — соавтор книги «Эффективные алгоритмы и сложность вычислений», автор научных и инженерных статей.

В 1995-1999 годах Стас Фомин был разработчиком информационных систем в компании «Сервер».

Стас Фомин закончил в 1999 году с отличием факультет управления и прикладной математики (ФУПМ) Московского физико-технического института.

# Reduce Project risk through early defect detection

Andreas Golze  
HP

## Abstract

1. Software defects are one of the biggest concerns for most companies who are using software as the backbone for their day to day operations.
2. Professional testing starting during the early design phase represents one of the biggest cost savings potentials for most organizations.
3. IT Governance driven through standardized quality assurance processes can be used a foundation to drive improvements of the software lifecycle.



**Bio:** Andreas Golze, Global Practice Director in the Professional Services department at HP (formerly Mercury), is an expert with many years` experience in the IT industry. He is one of the founders of the HP Software Quality Model, which is now the standard for many large-scale enterprise organizations around the globe. Before his move to HP, Andreas served as the Senior

Executive at the TestLab Business Unit of SQS AG, having earlier crafted and built up this unit in the role of Department Manager. He was also responsible for developing and expanding partner business. Andreas Golze studied computer science and minored in business studies at the University of the Armed Forces in Munich. His career began with a position as a programming officer at NATO in Belgium. Today, Mr. Golze lives in Cologne, Germany with his wife, Petra and their four children.



# Сокращение проектных рисков посредством раннего обнаружения дефектов

Андреас Гольце  
НР

## Тезисы

1. Дефекты программного обеспечения - одна из самых больших проблем в большинстве компаний, использование программного обеспечения в которых является основой для их повседневных операций.
2. Профессиональное тестирование, начинающееся во время ранней фазы проекта, представляет один из самых больших потенциалов снижения издержек для большинства организаций.
3. Стандартизованный процесс управления качеством может использоваться как основа для совершенствования жизненного цикла ПО.

**Биография:** Андреас Гольце, директор глобальной практики департамента профессиональных сервисов НР, эксперт в ИТ-индустрии с многолетним опытом. Андреас Гольце - один из основателей модели качества программных продуктов НР, которая является стандартом для крупномасштабных предприятий по всему миру. До своего прихода в НР, Андреас был руководителем тестовых лабораторий в SQS AG, также отвечал за развитие партнерских отношений.

Помимо основного изучения информационных технологий Андреас получил бизнес образование в университете вооруженных сил в Мюнхене и начал свою карьеру в Бельгии, в должности программиста в НАТО. Сейчас Андреас Гольце живет в Германии, в г. Кельне со своей женой Петрой и четырьмя детьми.

# **CMMI® for Acquisition: new model - for whom and why?**

**Alexander Kondakov**  
**“Inspirex Consulting” Consultant, Partner**  
**SEI Authorized Instructor**  
**Candidate SCAMPISM Lead Appraiser**  
**email: alexander@kondakov.ru**

## **Abstract**

The most habitual for the IT industry is the perception of model CMMI® as a set of the practices, recommended for application only in development of the software and systems. Model CMMI® initially included also some recommendations (in form of practices) related to management of suppliers and subcontractors. For the organizations where acquisition of services and the products prevails, such practices were not sufficient for the organization of processes. That's why "appendices" to the base model earlier were issued, with practices adaptation for such organizations. According to the concept developed at the moment of issuing of version 1.2 of model CMMI®, practices related to management of acquisition of products and services in the expanded variant have been placed in the separate model which has received name CMMI® for Acquisition (in abbreviated form CMMI-ACQ), officially published in November, 2007 (see [1]). In the proposed presentation the following themes will be covered:

- «target audience» of CMMI-ACQ model;
- linkage between CMMI-ACQ model and earlier published CMMI for Development (CMMI-DEV) model;
- differences between CMMI-ACQ и CMMI-DEV models;
- features of CMMI-ACQ model;
- commonalities of appraisal methods for CMMI-ACQ и CMMI-DEV model;
- sources of additional information about CMMI-ACQ model.

**Keywords:** CMMI model; CMMI-ACQ; CMMI models.

## **References:**

[1] CMMI for Acquisition, Version 1.2, CMU/SEI-2007-TR-017, CMMI Product Team

# **CMMI® for Acquisition: новая модель - для кого и зачем?**

**Александр Кондаков**  
**“Inspirex Consulting” Консультант, Партнер**  
**SEI Authorized Instructor**  
**Candidate SCAMPISM Lead Appraiser**  
**email: alexander@kondakov.ru**

## **Тезисы**

Наиболее привычным для индустрии информационных технологий является восприятие модели CMMI® в качестве набора практик, рекомендуемых к применению только при разработке программного обеспечения и систем. При этом модель CMMI® изначально включала и некоторые практики, связанные с управлением взаимодействием с поставщиками и субподрядчиками. Для организаций, в деятельности которых преобладает приобретение услуг и продуктов, указанных практик не хватало для организации процессов. В связи с этим ранее выпускались «приложения» к базовой модели, адаптирующие все практики для такого рода организаций. В соответствии с концепцией, разработанной при выходе версии 1.2 модели CMMI®, практики, связанные управлением приобретений продуктов и услуг в расширенном варианте были вынесены в отдельную модель, получившую название CMMI® for Acquisition (сокращенно CMMI-ACQ), официально опубликованную в ноябре 2007 года (см. [1]). В предлагаемом докладе будут рассмотрены следующие темы:

- «целевая аудитория» модели CMMI-ACQ;
- связь модели CMMI-ACQ и ранее вышедшей модели CMMI for Development (CMMI-DEV);
- различия моделей CMMI-ACQ и CMMI-DEV;
- особенности модели CMMI-ACQ;
- общность методов оцениваний относительно моделей CMMI-ACQ и CMMI-DEV;
- источники дополнительной информации о модели CMMI-ACQ.

**Keywords:** CMMI model; CMMI-ACQ; CMMI models.

## **Ссылки:**

[1] CMMI for Acquisition, Version 1.2, CMU/SEI-2007-TR-017, CMMI Product Team

# **Pitfalls in C#**

**Gaidar Magdanurov**  
**Microsoft**

## **Abstract**

Have you ever spent a sleepless nights debugging you application code and met unexpected behavior of C# code or standard libraries?

Some details on common “pitfalls“ developers frequently fell into are covered in this session. Real code samples in C# are used to demonstrate program execution results unexpected by most of developers. Each problem is covered in full details on behavior and some general ideas may be applicable to programming languages other than C#.

The primary goal of the session if to tease developers to apply out-of-box thinking to C# language.

## **Подводные камни C#**

**Гайдар Магдануров  
Microsoft**

### **Тезисы**

Приходилось ли вам проводить бессонные ночи в поиске ошибок в ваших программах, связанных с неожиданным поведением C# кода или стандартных библиотек?

В докладе рассказывается о некоторых «черных дырах», в которые часто попадают разработчики. Проблемы, с которыми разработчики не планируют сталкиваться при выполнении приложений, демонстрируются на распространенных примерах исходных кодов. Каждая проблема детально рассматривается в призма поведения приложения, ведь в большинстве случаев аналогичное решение может быть применено при написании кода на других языках программирования.

Цель доклада – предложить разработчикам по-новому взглянуть на привычные средства языка и возможности стандартных библиотек.

# **F#: Multiparadigm Programming at Industrial Scale**

**Dmitry Soshnikov, Academic Developer Evangelist**  
**Microsoft Russia**  
**email: [dmitryso@microsoft.com](mailto:dmitryso@microsoft.com)**

## **Abstract**

For many years, imperative programming has been the dominating paradigm in the industrial software development. Other paradigms, including functional and logic programming, have been used primarily by computer scientists in their research projects, but for some reason have not been adopted by the industry.

Things are beginning to change now, as more and more multiparadigm features are being introduced into mainstream programming languages. Speaking of .NET family of languages, functional features have been introduced into C# 3.0, including lambda-expressions, type inference, closures and LINQ data access mechanism, which allows us to argue that C# is becoming a multiparadigm language. However, it still remains primarily an imperative language, lacking the concise syntax that is typical for functional languages, and not enforcing such features as data immutability, etc.

The reason other programming paradigms are becoming increasingly important in modern software development is the increasing complexity of applications, including applications for multicore/manycore systems. Paradigms that employ immutable data structures allow for inherently parallel algorithms, thus greatly simplifying the job of creating multi-threaded applications for parallel computations. In addition, functional paradigm allows for more flexible functional abstraction techniques, provides more error-free code that require less debugging, and yields more compact code.

Taking the increasing importance of functional programming languages into account, Microsoft has plans to introduce a functional programming language, F#, into the next version of Visual Studio. F# is in fact a multiparadigm language, with some imperative features like mutable structures and loops, but primarily a functional one, derived from OCaml and ML family. It is fully interoperable with .NET framework, strongly statically typed language, having very concise syntax due to type inference. With those features, F# is becoming an attractive option for both

educational and research scenarios, as well as for more industrial applications and as high-level scripting language.

This presentation introduces the F# language, demonstrates some examples of using the language together with .NET framework, and discusses usage of functional programming in some industrial contexts, as well as some software engineering aspects of using functional languages in real-life projects.

**Keywords:** functional programming, F#, multiparadigm languages.

## **F#: Мультипарадигмальное программирование в индустриальном масштабе**

**Сошников Дмитрий Валерьевич**

**к.ф.-м.н., доцент**

**департамент стратегических технологий, Майкрософт**

**Россия**

**email: dmitryso@microsoft.com**

### **Тезисы**

Многие годы доминирующей парадигмой при создании промышленных программных систем было императивное программирование. Другие парадигмы, такие, как логическое и функциональное программирование, в основном использовались в исследовательских проектах и обучении, а в индустриальном масштабе – скорее в виде исключения.

В настоящее время ситуация начинает меняться, и в индустриальных языках программирования появляется все больше мультипарадигмальных элементов. Рассматривая семейство языков на платформе .NET можно заметить, что в языке C# 3.0 появилось множество функциональных элементов, таких, как лямбда-выражения, вывод типов, замыкания и встроенный язык запросов LINQ. Это позволяет причислить C# к мультипарадигмальным языкам программирования, хотя в большей степени он остается императивным объектно-ориентированным языком, не предоставляя достаточное количество стимулов для написания кода в функциональном стиле (неизменяемые данные, лаконичный синтаксис и т.д.).

Причиной возросшего интереса к другим парадигмам программирования в области индустрии программного обеспечения является возрастающая сложность приложений, в том числе многопоточных приложений для многоядерных процессоров. Функциональное программирование с неизменяемыми структурами данных требует от программиста соответствующего подхода к решению задач, позволяющего производить более эффективное распараллеливание алгоритма, упрощая тем самым параллельное программирование. В дополнение к этому, функциональная парадигма обеспечивает дополнительные возможности использования функциональной абстракции, приводит к более надежному, свободному от ошибок и к более компактному программному коду.



Понимая возрастающую важность функционального подхода, Майкрософт планирует включить в следующую версию Visual Studio в качестве базового языка функциональный язык программирования F#. Будучи мультипарадигмальным языком, с такими императивными возможностями, как циклы и модифицируемые структуры данных, F# обеспечивает сквозное двустороннее взаимодействие с платформой .NET, позволяя прозрачно использовать .NET Framework из функционального кода, а также подключать к императивным проектам библиотеки, реализованные на F#. В своей основе F# тем не менее является функциональным языком, наследником OCaml и семейства ML-языков, с лаконичным синтаксисом и статической типизацией с выводом типов. За счет краткости синтаксиса и наличия псевдо-интерпретатора F# становится привлекательной опцией не только в образовательных или исследовательских сценариях, но и при разработке реальных программных систем или в качестве скриптового языка. В данной презентации мы знакомим слушателей с языком F#, показываем несколько примеров использования языка совместно с .NET Framework (включая managed DirectX), а также обсуждаем вопросы использования функциональных языков в индустриальном программировании.

**Keywords:** F#, функциональное программирование, мультипарадигмальное программирование.

# **Evolutionary Design**

**Denis Miller,  
Agile Consulting  
dmiller@agileconsulting.ru**

## **Abstract**

Evolutionary design is an integral part of development. The foundation of evolutionary design is being formed for a few decades: the principles of agile, refactoring, simple design, development through testing (TDD), etc. Agile approach showed the second part of the evolutionary design - the human factor.

Support and development of design directly linked to the collective knowledge of the system and the principles on which it is built. Knowledge of the current system creates a theory of a particular project. Agile approach synchronizes the theory about the project among participants, seeking not only effective implementation of the current challenges, but also creates the possibility of the evolution of the system according to the emerging needs of the client.

The report will raise questions of relation of design and programming, documentation and communication in the team. It will be shown that the evolution of design project is directly related to the project team development and the establishment of a culture of team development.

**Keywords:** Agile, evolutionary design, design principles, refactoring, team management, patterns language

# Эволюционный дизайн

Денис Миллер,  
Agile Consulting  
dmiller@agileconsulting.ru

## Тезисы

Эволюционный дизайн является неотъемлемой частью развития системы. Несколько десятилетий формируются основы эволюционного дизайна: принципы гибкой разработки, рефакторинга, простого дизайна, разработки через тестирование (TDD) и др.

Agile подход показал вторую составляющую эволюционного дизайна – человеческий фактор. Поддержка и развитие дизайна системы напрямую связаны с коллективным знанием о системе и принципах, на которых она построена. Знание текущей системы создаёт теорию конкретного проекта. Agile подход синхронизирует теорию о проекте между участниками, добиваясь не только эффективного выполнения текущих задач, но и создаёт возможности эволюции системы согласно возникающих потребностей клиента.

В докладе будут подниматься вопросы соотношения проектирования и программирования, документирования и коммуникации в команде. Будет показано, как эволюция дизайна проекта напрямую связана с развитием проектной команды и созданием командной культуры разработки.

**Keywords:** Agile, эволюционный дизайн, принципы проектирования, рефакторинг, управление командой, patterns language

## 1. Введение

Ключевым моментом разработки программного обеспечения является проектирование. Принято считать, что без наличия серьезно проработанной архитектуры проект не передаётся в разработку. Насколько такой подход эффективен? Какова роль команды? Кто собирает лавры за успех проекта, а кто виноват в неудаче? Кто ответственен за архитектуру? Как и куда должна меняться архитектура?

Адаптивные (Agile) методологии разработки коммерческих приложений отвечают на поставленные вопросы. Ядро этого подхода - это человеческий фактор. А главный инструмент - эволюционный дизайн. За кажущейся простотой скрыто

сложное взаимодействие людей, множества техник и принципов.

## **2. Классический подход**

Как решает задачу проектирования и развития продукта классический (плановый) подход, основанный на модульной разработке приложений. Вопрос эволюции дизайна системы здесь не поднимается. Архитектура определяется на стадии проектирования, а дальше дело за реализацией. Потребности в изменениях и адаптации должны быть продуманы на стадии проектирования. Эволюция системы отсутствует, так как разработка является плановой и рассчитанной заранее. Эволюцией становится плановое наращивание ожидаемой и запланированной на стадии проектирования функциональности.

Плановость порождает необходимость ролей, таких как руководитель проекта, аналитик, архитектор, разработчики и тестировщики. Каждый из участников знает точно, что ему сделать и когда. Всё прогнозируемо, как в строительстве зданий и мостов, откуда был заимствован плановый подход разработки.

Но возникают сложности, кто-то их признает, кто-то закрывает глаза и занимается самообманом. В строительстве зданий конечный продукт является материальной вещью и этапы его создания легко декомпозируются и не изменяются. Но в индустрии разработки программного обеспечения конечный продукт обладает меньшей материальностью.

Компонентный подход, заимствованный из строительства зданий, позволяет провести декомпозицию системы на узлы/уровни. Совокупность узлов может быть запланирована, назначены ответственные за реализацию и остаётся только ждать, когда проект «созреет». Последовательно реализуя модуль за модулем, слой за слоем, словно возводится здание, так и программное обеспечение приобретает завершённый вид. К сожалению, здание программного обеспечения в голове клиента оказывается настолько индивидуальным, что клиент чаще только во время строительства начинает понимать что он хочет. В итоге, метафора «строительства зданий» даёт сбой,

цена разработки начинает расти, появляется текучка кадров в проекте, недовольство заказчика растёт.

Как же быть в этой ситуации? Первый вариант – **защитить** себя от изменений, придумать сложный процесс верификации на каждом этапе и процесс внесения изменений, задокументировать как можно больше и тщательней или ещё что-нибудь.

Будет ли это выходом?

### 3. Адаптивный подход

Современное общество разработчиков предложило изменить ряд ключевых моментов классической парадигмы «строительства».

Если система сложна, и постоянно пытается разойтись по швам. Если клиенты меняют требования. Если мир коммерческих приложений очень сильно зависит от рынка и коммерческой конъюнктуры. Может не стоит так ориентироваться на замораживание требований в начале разработки и строить тяжёлое монолитное здание?

Если первоначальная метафора строительства зданий в мире меняющихся требований не поспевает за клиентом, может быть стоит её изменить? Кстати, к созданию метафоры «строительства» приложились Кент Бек и Ворд Каннингем в далекие 1980-е годы; они же через 15 лет познакомили мир с Agile манифестом [1].

Может не следует защищаться от изменяющихся требований, а поставить их во главу угла? Поэтому подходит метафора **«создание программного обеспечения – это биржа»**. Скачки котировок сродни изменяющимся требованиям. Если мы ориентируемся на изменяющиеся требования, то мы обязаны чаще встречаться с клиентом и использовать принципы эволюционного дизайна, чтобы каждый раз адаптировать архитектуру под новые требования.

Появляется термин итерационности, который подменяет плановое следование вехам проекта. Каждая итерация становится маленьким водопадом. Продолжительность итерации от 2 недель. Короткий срок итерации вызывает

огромный пласт проблем: как организовать сбор требований, как проектировать, как поддерживать архитектуру гибкой, но легковесной, как распределять роли в команде. Классическое понимание указанных активностей изменяется. Но остаётся вопрос – что же является архитектурой проекта?

Ответ на вопрос такой: проектом становится совокупность код и знаний команды. А в некоторых проектах к этому прикладывается ещё описание проекта в виде предпроектной документации в виде специально подготовленных требований (истории пользователей), фотографий досок с обсуждениями и диаграмм на салфетках.

А что же такое архитектура? Под архитектурой понимается структура модулей плюс общее видение каждого участника проекта принципов и правил разработки проекта (=кода), которыми он руководствуется во время обсуждений и написания кода. Архитектурой становится некое ментальное знание разработчиков, которое позволяет принимать решения, отвечать о возможности или невозможности реализовать тот или иной функционал и усилиях, которые нужно предпринять. Архитектура становится знанием команды, некой теорией разработки конкретного проекта. А если архитектура это теория, то каждая теория должна передаваться с использованием своего языка. Так математики разговаривают на языке формул, лирики на литературном языке. Даже подростки на площадке обладают своим языком.

Каждый проект формирует свой собственный язык (или диалект) проекта, который образуется в начале проекта и постепенно развивается, эволюционирует. Носители (разработчики) языка получая опыт работы в предметной области обогащают его своими терминами (решениями), или решениями, которые повторно использовали в других проектах. Как и появление новых модулей вводит в ежедневное общение разработчиков новые термины (например, может появиться слово «калькулятор», а смысл его будет в появлении нового сервиса рассчитывающего критические данные). Так же могут уходить некоторые термины и понятия, например, слой «мэпперов» (слой, отвечающий за сохранение данных

приложения в базу данных) заменится на hibernate (библиотека работы с базами данных).

Понятие языка проектирования [2] для разработчиков была заложена Кентом Бекон и Ворд Каннингемом (создатели Extreme Programming) в 1987 году с введением понятия **Patterns Language**. В дальнейшем идею поддержали многие авторы. Так появились различные каталоги: шаблоны проектирования (design patterns) [3], архитектурные паттерны [4], паттерны предметных областей и др.

#### 4. Эволюция проекта

Эволюция проекта отражается на структуре модулей, классов и их взаимодействии. Но это вторичное проявление, изменение принципов и правил конкретного проекта первичны. Правила и принципы создают теорию и язык конкретного проекта. Если эволюция кода уже неоднократно рассматривалась как часть эволюционного проектирования, то языку и теории проекта отдавалось меньшее внимание.

Не секрет, что для эволюции кода проекта нам предоставляют набор подходов: “Refactoring”[5], принципы гибкой разработки [6], Simple Design [7] и инструмент микродизайна Test-Driven Development [8].

Остаётся вопрос эволюции теории (знания) о проекте, которая находится в головах команды. Эволюция знания о проекте определяется трансформацией командных и индивидуальных знаний, ценностей, принципов разработки конкретного проекта, а так же шаблонов поведения и организации. Это составляет пирамиду логических уровней проекта:



В классическом подходе разработки командное знание останавливается на реализации, а ведущие разработчики доходят до уровня шаблонов (обобщённых решений конкретной реализации). Но остановка на этом уровне не позволяет стратегически подходить к эволюции проекта, так как этот уровень знания определяется только текущим решением (теорией проекта). Когда каждый разработчик владеет полной информацией по всем уровням и эти знания синхронизированы между участниками, то эволюция знания (теории) о проекте отражается на эволюции дизайна. Второе невозможно без первого.

Адаптивный подход предлагает для развития теории о проекте на всех логических уровнях наилучшее решение – самоорганизующуюся команду. Принципы и практики которой позволяют гармонично развивать знание о проекте всеми его участниками. Создается крепкая основа (командное знание) развитию (эволюции) проекта в направлениях, которые определяет клиент. Задумайтесь о том, как влияют такие практики на развитие знания о проекте: формирование общего видения, командное проектирование, постоянная коммуникация, доступность клиента, демонстрационные встречи, коллективное владение кодом, парное программирование, да и простые ежедневные летучки (stand-up meeting).

Но с другой стороны, такой подход предъявляет повышенные требования к самим разработчикам. Команда



должна развивать не только свою теорию, но и знать и использовать ведущие мировые наработки, идеи и готовности меняться.

Лучшие идеи передаются в шаблонах и практиках, которые адаптируются в каждой команде. Шаблоны проявляются на всех уровнях разработки, начиная от написания кода до коммуникационных взаимодействий и управления. Общая иерархия шаблонов:

- практики управления проектом;
- шаблоны построения команды;
- шаблоны личной организации труда;
- шаблоны предметной области;
- архитектурные шаблоны;
- шаблоны проектирования;
- шаблоны кодирования.

Каждый набор шаблонов так же содержит внутри себя упомянутые выше логические уровни. Например, шаблоны проектирования зиждутся на принципах повторного использования компонент, делегирования, использования интерфейсов и композиции. А главная ценность шаблонов проектирования – гибкость.

## **5. Заключение**

Эволюция программного обеспечения основывается на изменении теории о проекте. Agile подход – это не серебряная пуля. Но Agile придает значение построению общей теории. Развитие в команде знаний о проекте, стремление к использованию мирового опыта, развитие языка проекта в совокупности создают культуру разработки. Поэтому Филипп Крачтен, создатель RUP, сказал: «Agility is not a technology, science, or product but a culture».

Появление и поддержка культуры разработки порождает новые командные роли (например, scrum master). В командах появляются внешние тренера (коучер), которые помогают командам перейти к новым эффективным способам работы. А после квантового скачка в мир Agile многие удивляются

простоте, эффективности и адаптируемости к существующим процессам предлагаемых практик.

## **6. Ссылки**

- [1] <http://agilemanifesto.org>
- [2] <http://www.c2.com/cgi-bin/wiki?HistoryOfPatterns>
- [3] Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес Приемы объектно-ориентированного проектирования. Паттерны проектирования.
- [4] Buschmann F. Pattern-Oriented Software Architecture: A System of Patterns.
- [5] Фаулер М. Рефакторинг: улучшение существующего кода
- [6] Роберт К. Мартин. Быстрая разработка программ: принципы, примеры, практика
- [7] Extreme Programming Explained: Embrace Change, Second Edition By Kent Beck, Cynthia Andres
- [8] Кент Бек Экстремальное программирование: разработка через тестирование

# **Win March. Adaptive Project Management: Principles and Examples**

**Arkhipenkov Sergey**  
**email: sarkhipenkov@gmail.com**

## **Abstract**

Such important subject as management of a software developer team is considered in this report. It is known that working efficiency of software developers can vary tenfold. That's why the manager's task is to make reproducible the high efficiency of intellectual activity. Ways and means to solve this problem are in application of adaptive project management intended for study and amendment of characteristics and structure of the object of management, notably people and their cooperation. In this report the author submits seven management principles. By means of them a manager can ensure top team efficiency. Application of these principles is illustrated by examples. The author demonstrates that well-managed project can be successfully carried out by the ordinary developer team.

**Keywords:** team management, leadership, project management, adaptive management.

# **Марш победителей. Адаптивное управление проектом: принципы и примеры**

**Архипенков Сергей**  
**email: sarkhipenkov@gmail.com**

## **Тезисы**

В докладе рассмотрены важные вопросы руководства командой разработчиков программного обеспечения. Известно, что производительность программистов может отличаться в десятки раз. Задача руководителя - сделать воспроизводимой высокую эффективность интеллектуальной деятельности. Путь к решению этой задачи – применение методов адаптивного управления, направленных на изучение и изменение свойств и структуры объекта управления: людей и их взаимодействия. Представлены семь принципов адаптивного управления проектом, используя которые, руководитель может обеспечивать наивысшую производительность команды. Применение принципов иллюстрируется примерами. Автор показывает, что хорошо управляемый проект может быть успешно выполнен обычной командой разработчиков.

**Ключевые слова:** руководство командой; лидерство; управление проектом; адаптивное управление.

## **Введение. Классические методы управления не работают**

Уместно провести аналогию между классическими методами, применяемыми в системах автоматического управления летательными аппаратами, и подходами к управлению программными проектами.

«Как получится». Разомкнутая система управления. Например, провели kick-off meeting, поставили задачи разработчикам и пошли готовить банкет по поводу успешного завершения проекта. Аналогия: баллистический полет. Можно, но недалеко и неточно.

«Водопад». Жесткое управление с обратной связью. Расчет опорной траектории (план проекта), измерение отклонений,

коррекция и возврат на опорную траекторию. Лучше, но не эффективно.

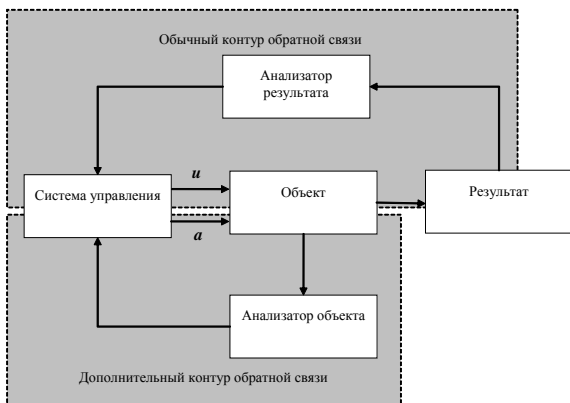
«Гибкое управление». Расчет опорной траектории, измерение отклонений, расчет новой попадающей траектории и коррекция для выхода на нее. «Планы - ничто, планирование - все» (Эйзенхауэр, Дуайт Дэвид)

«Метод частых поставок». Самонаведение. Расчет опорной траектории, измерение отклонений, уточнение цели, расчет новой попадающей траектории и коррекция для выхода на нее.

Классические методы управления перестают работать в случаях, когда структура и свойства управляемого объекта нам не известны и изменяются со временем. Эти подходы так же не помогут, если текущие свойства не позволяют объекту двигаться с требуемыми характеристиками. Например, летательный аппарат не может развить требуемое ускорение или разрушается при недопустимой перегрузке. Аналогично, если рабочая группа проекта не может обеспечить необходимую производительность и постоянно работает в режиме аврала, то это приводит к уходу профессионалов из проекта.



Рисунок 1).



$u$  – управляющее воздействие;  $a$  – адаптационное воздействие.

**Рисунок 1. Дополнительный контур обратной связи в системе адаптивного управления**

Результатом управления проектом являются: соответствие спецификациям, срок и бюджет.

Множество управляющих воздействий ( $u$ ) ограничено: составить план работ, расставить приоритеты, назначить на работы исполнителей. Как правило, таким и только таким управлением (администрированием) занимаются менеджеры - поклонники диаграмм Ганта и фанаты MS Project. Если у команды проекта низкая производительность, то единственный путь ее повысить при административном подходе - это постоянное давление, авралы, сверхурочные и субботники. Работать больше, это совсем не значит - работать продуктивнее. Скорее наоборот. Излишнее давление и суeta приводят к непродуманным решениям, большому проблемному коду и многочисленным последующим переработкам. Для подобных руководителей любой программный проект будет безнадежным.

Вместе с тем, известно, что производительность разных программистов может отличаться в десятки раз. Утверждаю, что производительность одного и того же программиста может так же отличаться в десятки раз. Заставьте лучшего в мире

бегуна бегать в мешке, и он покажет в 10 раз худший результат. Заставьте лучшего программиста заниматься «сизифовым трудом»: плодить документацию (которую, как правило, никто не читает) в угоду «Методологии» (именно с большой буквы 'М'), - и его производительность снизится в 10 раз.

Поэтому, основные усилия руководителя, если он стремится получить наивысшую производительность рабочей группы, должны быть направлены на изучение и изменение объекта управления: людей и их взаимодействия. Следовательно, задачу адаптивного управления мы можем разделить на две подзадачи:

1. Обеспечить эффективность каждого участника рабочей группы.
2. Обеспечить эффективные процессы взаимодействия.

Все люди разные и ситуаций, в которых они могут находиться в ходе проекта, бесчисленное множество. Бойтесь стереотипов. Если вы не учитываете индивидуальные особенности конкретной личности, то эффективность ваших взаимодействий сильно снижается.

Модель объекта управления нам неизвестна, следовательно, не может существовать исчерпывающий набор правил, тип «если..., то...», по которым смог бы действовать руководитель. Поэтому, сколько людей и ситуаций, столько и вариантов решений должен иметь эффективный руководитель в своем запасе. «Если у руководителя в руках только молоток, то все вокруг будут похожи на гвозди».

**Принцип 1.** «Принцип достаточного разнообразия». Для «хорошего» управления количество возможных состояний управляющего устройства (разнообразие) должно быть не меньше, чем количество состояний объекта управления [1].

Руководитель при поиске решения опирается на свой багаж знаний и умений. Он пытается понять каждого участника, классифицировать состояние, найти в своем опыте похожую ситуацию и адаптировать ранее использованное успешное

решение применительно к данному конкретному случаю. Таким образом, руководитель стремится помочь человеку (объекту управления) перейти в новое более эффективное с точки зрения целей проекта состояние.

Затем руководитель должен наблюдать за результатами своего воздействия – это и есть дополнительный контур обратной связи. Необходимо помнить, что понять человека можно, только слушая и слыша, что он говорит. Руководитель, который в течение недели не пообщался индивидуально с каждым из своих прямых подчиненных, зря получает зарплату. И совсем не обязательно разговор должен идти о статусе проектных работ. Порой, достаточно поговорить о погоде, кино или футболе.

После этого руководитель анализирует полученные результаты и аккумулирует новый опыт (положительный или отрицательный) в своей «базе знаний». Как это может происходить на практике, будет далее проиллюстрировано на примерах.

Чем опытней руководитель, тем точнее он может распознать и классифицировать сложившуюся ситуацию, тем больше в его «базе знаний» прецедентов, используя которые, он может синтезировать решение для данного конкретного случая. Именно поэтому в управлении программными проектами в первую очередь ценится опыт руководителя и только потом, возможно, его звание и знания.

### **1-я задача адаптивного управления. Обеспечить эффективность каждого участника**

Для того чтобы ваш сотрудник мог эффективно решить поставленную вами задачу, необходимо и достаточно выполнение четырех условий.

**Принцип 2.** «4 условия эффективной работы».

1. Понимание целей работы.
2. Умение ее делать.
3. Возможность ее сделать.
4. Желание ее сделать.



Для того чтобы обеспечить выполнение этих условий, руководитель должен уметь эффективно выполнять четыре функции.

**Принцип 3. «4 функции руководителя».**

1. Направлять. Если сотрудник не понимает что делать, задача руководителя - обеспечить общее видение целей и стратегии их достижения.
2. Обучать. Если сотрудник не умеет, задача руководителя – «обучать», быть наставником и образцом для подражания.
3. Помогать. Если у сотрудника не может выполнить работу, задача руководителя – «помогать», обеспечить исполнителя всем необходимым, убрать препятствия с его пути.
4. Вдохновлять. Если у сотрудника не достаточно желание выполнить работу, задача руководителя – «вдохновить», обеспечить адекватную мотивацию участника на протяжении всего проекта.

**Пример. «Хочет и может, но не делает»**

**Ситуация.** Программист стремится найти наиболее общее решение задачи, учесть все возможные последующие изменения и расширения. Старается разработать самый быстрый алгоритм, требующий минимальных ресурсов. Использует в решении все лучшие практики, паттерны проектирования, самые новые инструменты.

**Классификация.** Неоправданное усложнение задачи. Программист неадекватно понимает цели проекта и приоритеты. В результате стремления к совершенству - низкая производительность. Для выполнения работы в срок постоянно не хватает времени.

**Решение.** Направлять. Более четко формулировать цели и расставлять приоритеты. Определить конкретные критерии оценки качества результата. Нацеливать программиста на

правильное решение задачи максимально простым способом. Опыт свидетельствует, что более чем в 90% случаев ее не придется переделывать.

Пример. «Хочет, но не может»

Ситуация. Программист своевременно приходит на работу. Не отвлекается. Настойчиво работает над решением поставленных задач. Часто задерживается, чтобы уложиться в срок. В результате сроки, которые он сам оценивает, постоянно срываются. Большой проблемный код.

Классификация. У программиста недостаточно опыта. Он не умеет оценивать и планировать свою работу.

Решение. Учить самому или закрепить за программистом более опытного наставника.

Известно, что ни одна задача не будет решена за любое, отведенное на это время, если человек не захочет ее сделать. Он всегда найдет для оправдания этого 100 «объективных» причин, вместо того, чтобы найти хотя бы одну возможность для решения задачи.

У каждого участника рабочей группы должна быть личная цель (внутренняя мотивация), которую он сможет достичь, продвигая проект к успеху. Начните с себя! Вам нужно четко понимать, в чем состоит ваш выигрыш в случае успешного завершения проекта. Добиться от участников приверженности проекту больше, чем имеете вы сами, вам не удастся.

Если у участника нет такой личной значимой цели, избавьтесь от него. Иначе вам придется потратить все свое время на «промывание его мозгов» и попытки мотивировать его на эффективную работу.

Необходимо помнить, что по ходу проекта мотивы людей, как правило, изменяются.

Пример. «Может, но не хочет»

Ситуация. Программист имеет глубокие знания и развитый интеллект, быстро осваивает все новое, нацелен на решение

трудных задач. Пользуется заслуженным авторитетом среди коллег. В начале проекта активно выдвигал новые идеи, убедительно их обосновывал, добивался их признания всеми. Находил неизвестные возможности, существенно сократившие трудоемкость работ по проекту. В середине проекта потерял интерес. Стал «витать в облаках» и отвлекаться на изучение каких-то новых технологий. Постоянно заваливает сроки, делает глупые ошибки, непростительные для его опыта. Расхолаживающе воздействует на команду.

Классификация. Программист комфортно чувствует себя в роли «генератора идей» и не мотивирован на методичную реализацию.

Решение. Мотивировать. Например, пообещать роль архитектора, если данный проект завершится успешно. Или поручить роль наставника, предоставить возможность передавать свои знания и умения менее опытным коллегам.

Пример. «Может, но не хочет 2»

Ситуация. Программист активен, самостоятелен, напорист. По любому вопросу имеет свое собственное мнение. Всегда стремится быть победителем в конфликтах. Часто оценивает других и указывает им на недостатки. Использует любой повод, чтобы продемонстрировать свое превосходство. Сильно переоценивает свой личный вклад в общее дело и поэтому считает, что он должен работать меньше, чем его «менее способные» коллеги.

Классификация. Человек – эгоист, достиг личной независимости, но неспособен к конструктивному взаимодействию. Вредно воздействует на команду. Самооценка, скорее всего, неадекватно завышенная.

Решение. Избавиться. Если нет такой возможности, найти для него четко специфицированную, изолированную задачу, которая не находится на критическом пути проекта. Иметь под рукой другого специалиста, который сможет решить эту задачу, когда потребуются.

## **2-я задача адаптивного управления. Обеспечить эффективные процессы взаимодействия**

В отрасли программостроения многие уже признали, что наиболее эффективные производственные процессы складываются в самоуправляемых и самоорганизующихся рабочих командах, для которых характерны ясность общих ценностей и целей, самоконтроль, взаимопомощь, взаимозаменяемость, коллективная ответственность за результаты труда, всемерное развитие и использование индивидуального и группового потенциалов.

Эффективные команды не образуются сами по себе, они кристаллизуются вокруг признанного лидера. Как не бывает лидеров без последователей, так и не бывает команд без лидеров. Поэтому первый шаг руководителя при создании эффективной команды - это стать лидером, вокруг которого сможет сплотиться рабочий коллектив.

**Принцип 4.** «Принцип лидерства». Руководителю программного проекта недостаточно быть хорошим управленцем, он должен стать признанным лидером.

Лидера нельзя назначить. Лидер должен быть признан коллективом. Чтобы руководитель получил признание в качестве лидера, необходимо выполнение следующих двух условий.

1. Признание коллективом профессиональной компетентности и превосходства руководителя.
2. Полное доверие коллектива к действиям и решениям руководителя, признание его исключительных человеческих качеств, убежденность в его честности, порядочности, вера в его искренность и добросовестность.

Для того чтобы получить признание, если, конечно, руководитель его объективно заслуживает, он должен использовать принцип номер 5.

**Принцип 5.** «4 стратегии лидера». Не существует одной лучшей стратегии руководства. В зависимости от готовности участников рабочей группы выполнять задания руководителя, он должен использовать одну из 4-х стратегий [2]:

1. S1. «Директивное управление». Руководитель говорит, указывает, направляет, устанавливает. Жесткое назначение работ, строгий контроль сроков и результатов.
2. S2. «Объяснения». Лидер "продает", объясняет, проясняет, убеждает. Сочетание директивного и коллективного управления. Объяснение своих решений.
3. S3. «Участие». Лидер участвует, поощряет, сотрудничает, проявляет преданность. Приоритетное коллективное принятие решений, обмен идеями, поддержка инициативы подчиненных.
4. S4. «Делегирование». Лидер делегирует, наблюдает, обслуживает. «Не мешать» - пассивное управление сформировавшегося лидера.

**Пример.** «Ситуационное лидерство»

1. Ситуация. Вас назначили руководителем в новый коллектив. Вы еще не получили признания, а дело делать надо. Решение. Стратегия S1. «Директивное управление».
2. Ситуация. Вы были участником команды. Вас назначили руководителем этой команды. Доверие есть, а уверенности в правильности ваших действий нет. Решение. Стратегия S2. «Объяснения».
3. Ситуация. Вас назначили руководителем в новый коллектив. Все знают о ваших прежних сложных и успешных проектах. Все признают ваше превосходство, но доверия к вам нет. Никто не знает, какой ценой были достигнуты ваши победы. Решение. Стратегия S3. «Участие».

4. Ситуация. Между вами и участниками установлено взаимное доверие. Все достаточно мотивированы на успех проекта. Каждый сам себе может быть руководителем. Решение. Стратегия S4. «Делегирование».

Мало стать лидером, надо еще суметь сплотить коллектив. Эксперты в области командного менеджмента выделяют 4 обязательные последовательные стадии, через которые должна пройти рабочая группа прежде, чем она станет эффективной командой, это:

1. Forming. Формирование. Характеризуется избытком энтузиазма, связанного с новизной. Люди должны преодолеть внутренние противоречия, переболеть конфликтами прежде, чем сформируется действительно спаянный коллектив.
2. Storming. Разногласия и конфликты. Самый сложный и опасный период. Мотивация новизны уже исчезла, а сильные и глубокие стимулы у команды еще не появились. Неизбежные сложности или неудачи порождают конфликты и «поиск виновных». Участники команды методом проб и ошибок вырабатывают наиболее эффективные процессы взаимодействия.
3. Norming. Становление. В команде растет доверие, люди начинают замечать в коллегах не только проблемные, но и сильные стороны. Закрепляются и оттачиваются наиболее эффективные процессы взаимодействия. На смену битве амбиций приходит продуктивное сотрудничество. Четче становится разделение труда, исчезает дублирование функций.
4. Performing. Отдача. Команда работает эффективно, высок командный дух, люди хорошо знают друг друга и умеют использовать сильные стороны коллег. Все стремятся придерживаться выработанных общих

процессов. Высокий уровень доверия. Это лучший период для раскрытия индивидуальных талантов.

Часто случается, что рабочая группа вязнет на одной из стадий и никогда не достигает плато наивысшей производительности.

Пример. «Шумиха»

Ситуация. Частые смены приоритетов задач. Споры о том, что надо делать, а что не надо. Сомнения в реальности сроков. Недовольство отсутствием прогресса. Много вопросов по каждой задаче.

Классификация. Команда находится на первом этапе образования – «Объединение». Цели и стратегия их достижения не ясны и не приняты всеми участниками команды.

Решение. Стратегия S1. «Директивное управление». Функции: «направлять» и «помогать». Четко ставить цели и формулировать стратегию их достижения. Устанавливать проектные процедуры распределять роли и разграничивать зоны ответственности.

Пример. «Споры и дискуссии»

Ситуация. На совещаниях бесконечные неконструктивные споры и дискуссии. Постоянно доминируют одни и те же лица. Другие предпочитают отмалчиваться. Мнения высказываются как объективные факты. «На самом деле эта задача решается так...!» Постоянно даются оценки. «Это все неправильно!» «Это все не важно!» Присутствует агрессия «Ты, просто, ничего не понимаешь!». «А ты разве не знаешь, что...!»

Классификация. Команда находится на стадии «Разногласия и конфликты». Отсутствие культуры эффективных коммуникаций. Неумение слушать и слышать.

Решение. Стратегия S2. «Объяснения». Функции: «обучать» и «помогать». Стать для команды образцом эффективного взаимодействия. Конфликты необходимо разрешать спокойно, терпеливо и тщательно.

Пример. «Группомыслие»

Ситуация. Девиз участников проекта: «Давайте работать, а не конфликтовать!» Все стараются избегать конфликтов и поддерживать согласие. Как правило, никто не спорит, все соглашается с мнением руководителя и следуют его указаниям. При возникновении трудных ситуаций, все ждут решения от руководителя. Редкие противоречия разрешаются общим голосованием.

Классификация. В психологии подобная ситуация называется «Парадокс Абилина». Люди принимают решения, основанные не на том, что они сами хотят, но на том, что они думают, что другие хотят. В результате получается, что каждый делает что-то, что никому на самом деле не нужно. Подобное избегание производственных конфликтов снижает здоровую интеллектуальную конкуренцию, ведет к шаблонности и застою.

Решение. Стратегия S3. «Участие». Функции «помогать» и «вдохновлять». Целенаправленно мотивировать инакомыслие и интеллектуальную конкуренцию. Например, назначать рецензентов – критиков, которые должны найти слабые стороны в решении, предлагаемом их коллегой для общего обсуждения. Или поручить двум разработчикам решить одну и ту же критичную для проекта задачу, используя разные подходы или технологии, а затем поручить сравнить и оценить полученные результаты третьему коллеге.

Пример. «Менеджер должен занимать очередь...»

Ситуация. Ни одно предлагаемое участником команды решение не принимается на веру. Все требуют факты для его обоснования. Активно анализируются возможные негативные последствия или упущенные возможности при принятии решения. Конфликты носят исключительно производственный характер. При решении конфликтов активно ищутся взаимовыгодные возможности. Говоря словами Тома ДеМарко, «менеджер проекта должен занимать очередь, чтобы покриковать сотрудника, не выполняющего свои обещания».

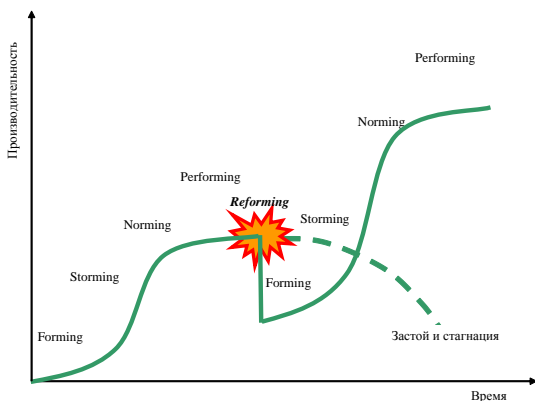


Классификация. Команда находится на стадии «Отдача». Наконец-то, она достигла плато наивысшей эффективности.

Решение. Стратегия S4. «Делегирование». Функции: «направлять» и «вдохновлять». Поддерживать требуемый уровень мотивации. Быть штурманом, искать новые пути и открывать новые возможности. Постоянно наблюдать и оценивать эффективность всех процессов, применяемых в проекте. Искать ответ на вопросы: «Что лишнее мы делаем?» «Что можно делать проще?» «Что угрожает проекту?». Работать на сокращение ненужных усилий вместо того, чтобы «стремиться к новым героическим подвигам».

Если руководитель не будет прилагать дополнительные усилия команда, рано или поздно, начнет «сползать» с плато наивысшей эффективности в состояние застоя и стагнации (Рисунок 2). Помните, что окружение и команда изменяются по ходу проекта. Прежняя мотивация ослабевает или перестает действовать.

**Принцип 6.** «Принцип цикличности». 4 стадии развития команды должны циклически повторяться, чтобы обеспечить непрерывный рост эффективности.



**Рисунок 2. Reforming. «Встряхивание» и перевод команды проекта на новый уровень производительности**

Изменяйте правила и процессы. Отказывайтесь от того, что перестало действовать или стало работать неэффективно. «Встряхивайте» (Reforming) и возвращайте команду в стадию Forming. Это позволит ей снова, пройдя через все этапы становления, выйти на новый более высокий уровень производительности.

Разумеется, делать это следует, после сдачи очередного релиза программного продукта, ну и, возможно, в случае глубокого кризиса проекта.

### **Заключение**

Ранее мы говорили, что результатом управления проектом являются: соответствие спецификациям, срок и бюджет. Утверждаю, что у успешного программного проекта должен быть еще один, четвертый результат: каждый участник команды уходил с работы в 18:00 с чувством победы. Чувство победы — это чувство удовольствия человека, который приближается к своей цели. А все, что человек делает с удовольствием, он делает максимально эффективно.

**Принцип 7.** «Принцип победителей». Программист состоит из четырех компонентов: тело, сердце, разум и душа.

1. Телу необходимы деньги и безопасность.
2. Сердцу - любовь и признание.
3. Разуму – развитие и самосовершенствование.
4. Душе – самореализация.

Предоставьте все это вашим сотрудникам, и эффективность их труда возрастет многократно.

В противном случае люди, которые хотят побеждать, найдут все это в другой команде, а в вашей останутся только неудачники.

#### **Ссылки**

- [1] У.Р.Эшби “Введение в кибернетику” М, ИЛ, 1959
- [2] Hersey P., Blanchard K.H. “Management of Organizational Behavior”, 6th ed., Englewood Cliffs: Prentice-Hall, 1993.

# **iCarnegie curricula remote delivery in Russia: the first results**

**Dr. S.V. Zykov, Ph.D.**

**TEKAMA Ltd.**

**4, 2nd Roshchinskaya St. 115191 Moscow Russia**

**E-mail: sergey.zykov@tekama.com**

## **Abstract**

Carnegie Mellon University is a leader of IT education in the U.S.A. Though Carnegie Mellon curriculum has been implemented in quite a number of major and rapidly developing IT countries, its educational model has not become well known in Russia yet. The purpose of the paper is to discuss the problems and results of the first Carnegie Mellon remote delivery implementation in Russia. Also, an innovative approach to remote delivery of the curriculum is presented, which is a pioneering experience world over.

**Keywords:** Software development, information technology, remote education.

# **Первые итоги дистанционного обучения по программе iCarnegie в РФ**

**С.В. Зыков, к.т.н., доц. ГУ-ВШЭ  
ООО «ТЕКАМА»**

**115191 РФ, Москва, 2<sup>я</sup> Рощинская ул, 4**

**E-mail: sergey.zykov@tekama.com**

## **Аннотация**

Университет Carnegie Mellon – ведущий вуз США в области информационных технологий. Несмотря на то, что программы этого учебного заведения реализованы и востребованы в целом ряде крупнейших и бурно развивающихся информационно-технологических держав мира, в России модель обучения пока не получила должной известности. Цель настоящей статьи – изложение проблем и результатов первого внедрения дистанционного обучения по программе Carnegie Mellon в нашей стране.

**Ключевые слова:** Разработка программного обеспечения, информационные технологии, дистанционное образование

## **1. Введение**

Университет Carnegie Mellon неоднократно признавался одним из лучших вузов США по направлению информационных технологий. К сожалению, в нашей стране этот вуз пока не достаточно широко известен. Настоящая работа ставит целью представление трудностей новаторского процесса внедрения курсов Carnegie Mellon в России, показывая первоначальные результаты проекта iCarnegie [1] – внедрения и адаптации Интернет-ориентированных курсов Software Systems Development (SSD). Кроме того, рассматривается подход к постановке дистанционного обучения по этой программе, предпринятый впервые в мире.

## **2. Особенности учебной модели iCarnegie**

Принципиально новая модель обучения включает тесную обратную связь с разработчиками курсов. Уникальность

обусловлена институтом менторов – наставников для преподавателей, а также кураторов подготовки курсов. При этом важной особенностью менторов, необходимой для корректного функционирования этой обратной связи, является тесное, непрерывное общение со студентами, где менторы выступают в роли (рядовых) преподавателей курса.

Другой интересной особенностью является постоянная динамическая подстройка содержания курсов под потребности аудитории. При таком подходе занятие состоит из нескольких (обычно 2-3) мини-лекций по 15-20 минут, перемежающихся, по мере необходимости, практическими занятиями для иллюстрации и более глубокого усваивания материала.

Таким образом, курсы находятся в состоянии непрерывной динамической адаптации к потребностям аудитории по нескольким параллельным «каналам» (инструкторы, менторы, разработчики, индустрия), что существенно улучшает качество знаний на выходе.

Высокая степень безопасности и надежности системы доставки и управления контентом курсов позволяет поддерживать адаптируемые пополнения учебных материалов (лекции, лабораторные упражнения, тестовые и экзаменационные вопросы и практические задания и др.) в режиме 24/7 с возможностью персонализированной коррекции содержания.

При этом важнейшие принципы образовательного подхода iCarnegie включают:

- тщательный подбор теоретических материалов (по мнению журнала US News, Университет Carnegie Mellon является «лучшей в США школой Computer Science»)[2];
- широкое использование Интернет-технологий;
- близость тематики и условия выполнения практических упражнений к реальной работе будущих выпускников.

### **3. Подготовка слушателей**

Важнейшим умением, приобретаемым студентами курсов (в ходе и по окончании обучения) становится склонность к метаобучению, что реализует известный принцип советской школы: «Учись учиться!».

Важно отметить, что целевая практическая направленность курсов приводит к доминированию практики над теорией. Так, временные рамки практических работ (лабораторные и

самостоятельные упражнения, задания, зачетно-экзаменационные испытания и др.) по меньшей мере вдвое превышают объем теоретической составляющей.

Ведущая «школа компьютерных наук» США хорошо понимает и строго дифференцирует теоретическую платформу computer science и прикладной аспект software engineering, отдавая в данном случае приоритет последнему, в силу необходимости использования результатов обучения непосредственно в бизнес-практике.

#### **4. Обеспечение качества**

В дополнение к качеству контента курсов, обусловленному ведущей ролью Университета в информационно-технологической (ИТ) отрасли США, особенности использования Интернет-среды дают такие преимущества, как:

- единая организация самодостаточных, легко корректируемых гетерогенных материалов (программное обучение (ПО) к курсам и инструкции по его установке, формы контроля успеваемости, мультимедиа-иллюстрации, исходные тексты лекций, библиография и т.д.) в форме (гипер)текста;
- постоянная доступность (24/7/365);
- легкость тиражирования материалов курсов, включая перевод на иностранные языки.

Высокие стандарты качества, предъявляемые университетом к программе, позволяют обеспечить стандартизацию терминологии, осуществить тщательный подбор материалов ведущих ИТ-авторов, строго связать основную и дополнительную литературу с разделами курсов (включая иноязычные и переводные издания).

Качество обучения (и ценность сертификатов программы iCarnegie) удастся обеспечить, благодаря практической ориентации выполняемых упражнений, умению охватить проблему в целом, не упустив существенных деталей, «жесткими» условиями тестирования (временными ограничениями и требованиями по проценту правильных ответов). Курсы четко ориентированы на конкретные должности, выпускников, содержат тщательно подобранные практические примеры. При этом важными требованиями к преподавателям являются необходимость адаптации к уровню студентов, использования передовых технологий,

инструментария, «практик», стандартов, а также решений, независимых от конкретного производителя ПО. Подобный подход делает курсы более экономичными, а их выпускников – более универсальными специалистами, готовыми сразу после выпуска к принятию обоснованных, взвешенных технико-технологических решений в сфере своей компетенции.

## **5. Применение Интернет-технологий**

Важнейшей идеологической составляющей учебного процесса является взаимодействие его участников с Интернет-средой и в Интернет-среде. Подобная ориентация обеспечивает возможности совместной проектной работы при создании программных продуктов в ходе освоения курсов, способствует общению студентов (и преподавателей), облегчает распространение курсов и формирование гибкой обратной связи. Последняя является важнейшим условием поддержания неизменно высокого уровня качества учебного процесса и его результатов.

Достижению перечисленных целей способствует созданная в университете программная среда TRESTLE, которая обеспечивает:

- систематизацию учебной информации в единой гипертекстовой базе данных
- надежное хранение персональных данных и быстрый поиск по ним
- ведение «классного журнала» с полной статистикой успеваемости
- возможность выполнения лабораторных работ, сдачи тестов и экзаменов
- безопасный и быстрый обмен информацией между студентами
- обратную связь с инструкторами и менторами
- В основе ПО TRESTLE лежат широко распространенные и инвариантные к Интернет-браузерам язык Javascript, а также технология session cookies.

## **6. Структура и подача материалов**

Курсы традиционно включают мини-лекции и более объемные практические занятия. Целью первых является объяснение основных принципов и подходов курса, вторые



нацелены на техника проектирования и реализации ПО и гибко адаптируемы к тому уровню навыков программирования, который имеется у студентов.

Общая структура курсов близка к традиционному вузовскому формату, т.к. требует осмысления промежуточных результатов и насыщена объемными практическими упражнениями.

Средняя продолжительность курсов составляет 10-12 недель или 40-50 академических часов. При этом рекомендуемая периодичность занятий составляет 2 раза в неделю по 3 академических часа. Формат занятия, с учетом адаптации к специфике российской аудитории имеет следующий вид: теория – два 20-минутных «слота» отводятся на теоретический материал, два 40-минутных «слота» посвящены практике. Другими важными формами работы в «семестре» являются интерактивное выполнение практических заданий в Интернет-среде и самостоятельная работа студентов. Их объем весьма значительный – рекомендуемый уровень составляет порядка 150 часов.

Материал курса разбит на «уроки» (unit), прохождение каждого из них, в зависимости от объема и сложности требует 1-3 недель.

Формы контроля успеваемости разнообразны и включают онлайн-опросные листы и экзамены по теоретическому материалу (в среднем – еженедельно), а также большое количество разнообразных практических задач и упражнений (в среднем по два в неделю, рекомендуемая продолжительность каждого – 1,5-2 часа).

Курсы завершаются сертификационным экзаменом который (как и промежуточные) содержит теоретическую и практическую части и сдается аудиторно, под строгим контролем преподавателя. Результаты экзаменов (как и промежуточных форм контроля) обязательно фиксируются в ПО TRESTLE.

## **7. Тематика учебных программ**

Рассматриваемые курсы, при колоссальных внешних различиях, имеют важное общее свойство – оба они посвящены подготовке специалистов в области инжиниринга ПО. Первый курс (SSD1 – «Введение в информационные системы») дает введение в данную проблематику, второй (SSD9 – «Разработка

спецификаций, тестирование и сопровождение ПО») является курсом уровня магистратуры и венчает обучение по специальности. Таким образом, уже курс SSD1 предлагает упрощенную методологию проектирования, реализации и тестирования ПО, практически приемлемую для небольших программных проектов, тогда как курс SSD9 рассматривает все основные процессы жизненного цикла ПО.

В этой связи «входные требования» к студентам SSD1 минимальны и составляют основные знания математики на уровне программы средней школы, тогда как SSD9 требует практического опыта работы в программных проектах и знания основ программирования, проектирования ПО и баз данных.

Тематика SSD1 включает знакомство с языком HTML и базовыми веб-технологиями (гипертекст и его форматирование, работа с таблицами и изображениями, изучение поисковых механизмов, создание статических веб-страниц и веб-форм), а также обучение азам программирования для Интернет с использованием технологий Java (среда J2SE, установка, компиляция, запуск, отладка программ, основные операторы, сервлеты и др.). При этом важным преимуществом курса перед аналогами является изучение принципов и разнообразных аспектов объектно-ориентированного подхода (наследование, создание и модификация классов, обмен сообщениями, использование средств автоматизации документирования, методологические основы проектирования и реализации объектных программ).

В результате слушатели получают знания (основ работы в Интернет, языка HTML, серверных компонент Java) , достаточные не только для создания простых и эффективных веб-сайтов, но и их современного динамического расширения, а также продуманной организации их документирования, сопровождения и поддержки.

Ключевыми для будущей работы и роста в сфере ИТ становятся знания основ процесса разработки ПО, важнейшие концепции ООП, постановка «хорошего» стиля и техники программирования, технологии и средств документирования программных проектов.

Как уже отмечалось, курс SSD9 является существенно более сложным для успешного прохождения и требует, по нашим оценкам, тщательного анализа опыта слушателей и личного собеседования с каждым из них.

Для демонстрации различий в уровне рассматриваемых курсов перечислим входные требования к студентам SSD9:

проектирование концептуальных и реляционных схем, реализация БД;

знание различных моделей жизненного цикла ПО, языков запросов, техники нормализации данных, выполнения транзакций, индексирования БД;

- владение технологиями построения Интернет-приложений, 3-уровневой архитектуры «клиент-сервер», Java, ООП, РСУБД;
- навыки настройки БД для обеспечения масштабируемых, эргономичных, отказоустойчивых, надежных, производительных приложений;
- знание базовых технологий и приемов проектирования, тестирования и реализации ПО с БД, построения SQL-запросов.

Важнейшими принципами курса являются ориентация на экономичные и надежные, современные и достаточно универсальные технологии и средства проектирования ПО, изучение полного жизненного цикла проекта – от создания требований до сопровождения. Напомним, что курс соответствует магистерскому уровню требований ведущего вуза по computer science в США. Заметим, что преподавание SSD9 возможно по двум схемам: индивидуальная и командная разработка (пока в России в силу новизны проекта применяется только первая схема).

Тематика курса включает обзор моделей жизненного цикла ПО, технологии анализа предметной области, методы построения проектных спецификаций, объектно-ориентированный подход к анализу и проектированию, а также методики сборки, тестирования, документирования, внедрения и сопровождения программных проектов.

В ходе обучения слушатели реализуют полный жизненный цикл учебного программного проекта – небольшой системы электронной коммерции для онлайн-торговли музыкальными инструментами. Проект в трехуровневой архитектуре «клиент-сервер» включает разработку визуального Java-интерфейса, работу с СУБД и средствами связи БД с интерфейсом (JDBC).

Многочисленные практические задания курса SSD9 требуют большого труда как со стороны студентов, так и от преподавателя (осуществляющего как компьютерный, так и

«ручной» контроль многостраничных заданий в форме эссе, UML-диаграмм и т.д.), и включают:

- спецификации проекта, требований, выбор модели проектирования;
- логическое и ER-моделирование предметной области, проектирование схемы РБД;
- распределение ролей в проекте;
- выделение первичных классов;
- UML-диаграммирование;
- «ответственное» проектирование;
- внедрение проекта, план тестирования;
- документирование;
- финальная демонстрация клиенту (инструктору);
- сопровождение.

Важными принципами, которые рекомендуются для усвоения принципиальных идей курса, являются, прежде всего, его «сквозной» характер, нацеленный на полный охват жизненного цикла проекта, а также воспитание должного уровня дисциплины проектирования (в т.ч. посредством «ответственного» проектирования (responsibility-driven design)).

Базовые технологии и средства, необходимые для освоения SSD9, включают различные виды диаграмм (ERD, DFD, STD, OOAD), технологии «клиент-сервер», Sun Java 2 с JSP и сервлетами, Интернет-технологии (Apache Tomcat), СУБД (mySQL, Microsoft Access, PostgreSQL), средства UML-диаграммирования (Microsoft Visio, Microsoft Word, EclipseUML), средства документирования (Javadoc), драйверы для связи с БД (ODBC/JDBC).

По завершении курса слушатели достигают уровня квалификации старшего разработчика (senior developer) проекта. Они овладевают приемами проектирования предсказуемого и экономичного ПО, критического анализа и руководства всеми стадиями его жизненного цикла (в т.ч. повторным использованием), систематизируют представления об архитектурах ПО, применяют технологии OOAD / CASE, умеют строить масштабируемое клиент-серверное ПО с БД-компонентами.

## 8. Дистанционное обучение по программе iCarnegie

Общие принципы проведения дистанционного обучения (ДО) сводятся к следующему. Одинаковая скорость работы студентов в семестре задается в начале семестра графиком прохождения тем и выполнения практики. Необходим одновременный (или близкий по времени) старт группы. Преимущественным каналом общения студентов и инструкторов является система сообщений ПО для обучения iCarnegie – TRESTLE. Время реакции инструктора на запрос студента в системе сообщений TRESTLE не должно превышать 1 суток, а время оценки инструктором практического задания студента в системе TRESTLE – 2 суток. Необходимо обеспечить еженедельную отчетность о ходе работы по курсу в установленном формате между всеми участниками ТЕКАМА, компаний-партнеров и iCarnegie.

ТЕКАМА осуществляет обучение по курсу в дистанционном формате за исключением (сертификационных) экзаменов. Сертификационные экзамены проводятся очно, в среде, соответствующей требованиям iCarnegie, под контролем ведущего инструктора/ментора (или, в его отсутствие, под контролем полностью сертифицированного инструктора) по курсу. Оценка сертификационного экзамена проводится ведущим инструктором/ментором или любым сертифицированным инструктором по курсу.

Промежуточные экзамены проводятся под контролем ассистента (без сертификации iCarnegie или опыта преподавания курса iCarnegie) в открытой среде партнерского учебного центра («Сетевая академия ЛАНИТ»).

Для контроля корректности проведения экзаменов используются видеокамеры и/или веб-камеры, направленные на каждого из студентов в ходе экзамена. Видеозаписи экзаменов передаются в ТЕКАМА и хранятся до утверждения и/или отказа в сертификации каждого из студентов. В ходе экзаменов необходимо обеспечить возможность оперативной связи с ведущим инструктором/ментором (через helpdesk в ТЕКАМА или по мобильной связи).

Инструктор/ведущий инструктор/ментор по курсу регулярно проводит удаленные сессии (в среднем 1,5-часовые, минимум 3 раза в неделю) для поддержки студентов в ходе ДО.

Проведение курсов на местах организуется на базе филиалов авторизованных учебных центров ЛАНИТ или корпоративных структур (при корпоративном обучении). Каждая из организаций-партнеров должна располагать компьютерным классом, соответствующим требованиям к курсам iCarnegie (включая аппаратное и программное обеспечение, видеокамеры, Интернет-канал с межсетевым экраном, справочная литература), а также персоналом технической поддержки, обученным ТЕКАМА для ДО по курсам iCarnegie.

Для повышения качества обучения среди студентов (через организации-партнеры) перед началом курса распространяется справочная литература (литература на русском языке – только из списка, согласованного с iCarnegie), а также презентационные материалы на русском языке. Кроме того, контент «младших» курсов SSD (SSD1-SSD5) переводится на русский язык и выверяется с iCarnegie до старта ДО по каждому из курсов. Контент «старших» курсов SSD (SSD6-SSD10) переводу на русский язык не подлежит (предоставляется студентам в оригинале). Обеспечение студентов справочной литературой и презентационными материалами к курсам является задачей компании ТЕКАМА (функция службы helpdesk).

Перед началом первого семестра ДО по каждому курсу компания ТЕКАМА командировывает ведущего инструктора/ментора в iCarnegie для прохождения менторской стажировки. По окончании менторской стажировки ведущий инструктор/ментор проводит обучение ассистентов, службы технической поддержки и helpdesk.

В случае отсутствия по уважительной причине (отпуск, командировка, болезнь и т.д.) ментора по курсу, его функции выполняет ведущий инструктор, в его отсутствие – рядовой инструктор.

Основные роли в команде по ДО (по каждому курсу SSD): ментор, ведущий инструктор, рядовой инструктор, ассистент инструктора, служба технической поддержки, служба helpdesk. Все роли, кроме ментора/ведущего инструктора курса, могут быть множественными.

В обязанности ментора курса входит:

- взаимодействие с ментором курса iCarnegie по обновлению контента курса, проблемам проекта ДО и др. (в т.ч. еженедельные отчеты);

- руководство ДО (в т.ч. контроль корректности установки ПО для сертификационных экзаменов, контроль доставки материалов по курсу – презентаций, инструкций по установке ПО, дополнительной литературы и т.д., ведение статистики и др.);
- ведение и обновление базы знаний по курсу (проблемы, статистика, ошибки/несоответствия в контенте/материалах курса, планирование обновления и развития контента/материалов курса и т.д.);
- обучение рядовых инструкторов (в т.ч. сложные для студентов места курса, особенности подготовки к экзаменам и др.);
- контроль и оценка сертификационных экзаменов;
- обучение служб технической поддержки и helpdesk в ТЕКАМА и организациях-партнерах;
- проведение ДО (дистанционные сессии, проверка заданий и т.д.).

Обязанности ведущего инструктора совпадают с обязанностями ментора, за исключением взаимодействия с ментором курса iCarnegie, а также включают выполнение обязанностей ментора курса в его отсутствие. Обязанности рядового инструктора совпадают с обязанностями ведущего инструктора, за исключением руководства ДО и ведения и обновления базы знаний по курсу, а также включают выполнение обязанностей ведущего инструктора в его отсутствие. Обязанности ассистента инструктора совпадают с обязанностями инструктора, а также включают выполнение обязанностей инструктора в его отсутствие.

В обязанности сотрудника технической поддержки входит:

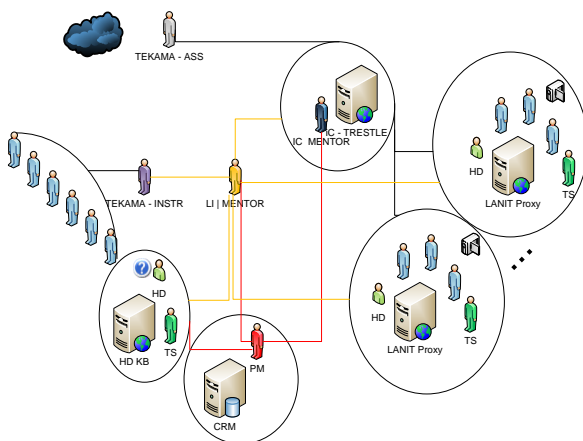
- подготовка класса для проведения экзаменов (в т.ч. АО, ПО, Интернет, межсетевой экран, видеокамеры, литература);
- техническая поддержка студентам/ ассистенту/ ведущему инструктору/ инструктору/ ментору в ходе экзаменов;
- оперативное информирование ассистента/ инструктора/ ментора о проблемах в ходе экзаменов.
- В обязанности сотрудника helpdesk входит:
- ведение расписания ДО (график онлайн-сессий, расписание сдачи заданий / экзаменов / статус сертификации в TRESTLE, график командировок / отпусков персонала, пропуск/перенос онлайн-сессий и т.д.)

- сбор информации (feedback) от студентов;
- прием, сортировка и направление заявок ассистенту/ведущему инструктору/инструктору/ментору;
- централизованное дублирование/передача информации студентам от ассистента/ведущего инструктора/инструктора/ментора (в т.ч. расписание экзаменов, поступление презентаций, инструкций, литературы, состояние АО и ПО на местах, TRESTLE, проблемы контента, доступ в Интернет);
- ведение базы знаний (контакты, часто задаваемые вопросы, текущие организационные опросы и др.);
- оперативное информирование helpdesk партнеров/ассистента/инструктора/ведущего инструктора/ментора о проблемах в ходе ДО.

Перекрытие обязанностей в команде ДО имеет целью взаимозаменяемость, масштабируемость и надежность процессов. Каналы связи между ТЕКАМА, организациями-партнерами и iCarnegie для организации ДО включают систему сообщений TRESTLE (приоритетно), систему сообщений MSN / Windows Live, электронную почту, службу Microsoft NetMeeting (входит в состав ОС Windows), средства мобильной и офисной телефонии (в т.ч. конференц-связь), а также средства факсимильной связи.

Схема организации дистанционного обучения по программе iCarnegie представлена на рис.1. Поясним обозначения на схеме: ТЕКАМА-ASS – ассистент инструктора, ТЕКАМА-INSTR – инструктор, LI – ведущий инструктор, MENTOR – ментор, HD – сотрудник helpdesk, TS – сотрудник службы технической поддержки, KB – база знаний, PM – руководитель программы, CRM – ПО управления отношениями с клиентами, LANIT – «Сетевая академия ЛАНИТ», iC – iCarnegie, Proху – прокси-сервер, контролирующий сдачу экзаменов в Интернет-среде.





**Рис. 1. Схема организации дистанционного обучения по программе iCarnegie.**

## **9. Результаты апробации**

В ходе апробации дистанционного обучения по курсам университета Carnegie Mellon в нашей стране были получены следующие практические результаты.

Завершены два семестра дистанционного обучения по начальному курсу SSD1 – «Введение в информационные системы». Практика апробации привела к следующим итогам.

Для продвинутых студентов оказалось вполне достаточной продолжительная вводная беседа в режиме телеконференции в течение 1 часа. Дальнейшее общение оказалось целесообразно построить по каналам электронной почты. В крайних случаях было решено применять также средства он-лайн общения (messaging), встроенные в систему аттестации TRESTLE, а также телефонные переговоры или телеконференции.

Выяснилось, что повышение оперативности взаимодействия со студентами при проверке выполненных заданий (в день сдачи, а не в течение 2-3 дней) может существенно (в среднем – на 25-50%) ускорить процесс учебный процесс для продвинутых студентов. При этом среднее время выполнения задания для таких студентов практически не отличается от такового для очной формы. Естественно, при условии сверхоперативной обратной связи (при которой проверка заданий и/или направление

преподавателем комментарии осуществляется не позднее, чем в день сдачи студентом этих заданий).

При интенсивном взаимодействии студентов и преподавателей крайне востребованным оказалось наличие резервного Интернет-канала, который потенциально позволяет ускорить прохождение курса более чем на 10-20%.

В ходе экспериментов было также выявлено, что продвинутые студенты практически не испытывают трудностей с установкой и настройкой программного обеспечения к начальным курсам. Сложности установки и настройки программного обеспечения к курсу определяются не формой обучения, а исключительно уровнем знаний и умений слушателя. В то же время, при старте курсов, требующих разнообразного программного обеспечения высокой сложности (скажем, СУБД, CASE- средств и др., как в курсе SSD9 – «Описание, тестирование и сопровождение программного обеспечения»).

Ряд мероприятий (и, прежде всего, промежуточные экзамены) в случае дистанционного обучения требует меньших затрат времени по сравнению с очной формой (ведь в последнем случае нужна специальная подготовка класса и выполнение ряда других предварительных операций, а также очный контроль со стороны преподавателя). При этом, естественно, необходимо соблюдение, по крайней мере, следующих условий:

1) тщательная проверка материалов для тестирования уровня знаний;

2) заблаговременная организация индивидуального допуска слушателей к испытаниям (для этого предусмотрены специальные процедуры в среде обучения TRESTLE).

## **10. Заключение**

В результате практического внедрения технологий университета Carnegie Mellon в нашей стране (с учетом их творческой адаптации) можно сделать следующие предварительные выводы.

Отдельные недостатки, которые характеризуют специфику российской аудитории и лишь подчеркивают достоинства курсов, сводятся к необходимости учета ориентации российской ИТ-отрасли на специфичное ПО, терминологию и работы западных авторов. Ряд перечисленных проблем, в

основном, удалось удовлетворительно решить при адаптации курсов: разработаны оригинальные презентации, сопроводительные материалы к лекциям, отобрано взаимоприемлемое ПО, согласована литература к курсам.

Курсы доработаны для оптимизации сочетания теории и практики с точки зрения реальной работы и бизнеса.

Сохранен традиционно высокий уровень качества, обеспечиваемый апробированной методикой подготовки студентов и оценки их знаний.

Важным преимуществом курсов является их четкая ориентация на конкретные обязанности и должности ИТ-специалистов.

Другие позитивные особенности курсов включают использование передовых технологий, инструментария, «практик» и стандартов, а также приоритетность технологий, не зависящих от производителя (что обеспечивает экономию и универсальность подхода).

Тщательность подбора материалов для изложения и оценки знаний слушателей подчеркивает постоянная обратная связь с индустрией, разработчиками курсов и вузами, на основе сбора и анализа колоссального объема многолетних статистических данных, включающих целый ряд параметров (в т.ч. влияние гендерного фактора на обучения, сдачу экзаменов и сертификацию).

В настоящее время ведется внедрение схемы дистанционного образования по программе iCarnegie в России и странах СНГ согласно технологической схеме, разработанной автором статьи.

## **Литература**

[1] iCarnegie Ltd.: <http://www.icarnegie.com>

[2] US News Colleges Ranking Review. [http://colleges.usnews.rankingsandreviews.com/usnews/edu/college/directory/brief/drglance\\_3242\\_brief.php](http://colleges.usnews.rankingsandreviews.com/usnews/edu/college/directory/brief/drglance_3242_brief.php)

[3] TEKAMA Ltd. <http://www.tekama.com>

[4] Зыков С.В. Пилотная постановка очных и дистанционных учебных курсов iCarnegie в России и странах СНГ // II Всероссийская научная конференция с международным

участием «Технологии информатизации профессиональной деятельности (в науке, образовании и промышленности)», 10-14 ноября 2008 г.– Ижевск: УдГУ, 2008 (готовится к печати).

# **CORRELATION BETWEEN CODING STANDARDS COMPLIANCE AND SOFTWARE QUALITY**

**Author: Wojciech Basalaj**

**Co-Author: Frank van den Beuken**

**Programming Research, 9-11 Queens Road, Hersham, Surrey  
KT12 5LU, UK**

**Frank\_van\_den\_Beuken@programmingresearch.com**

## **Abstract**

Software Quality has different meaning to different people. The ISO 9126 standard was developed to introduce clarity and establish a framework for quality to be measured. This paper aims to explore how Internal Quality characteristics of a software system (source code) can be measured effectively. Instead of relying on traditional software metrics, which are shown to be a poor predictor of underlying software quality, we advocate measuring compliance to a coding standard. We show qualitative and quantitative evidence of how adoption of a coding standard helps organizations in improving the quality of their C/C++ software.

**Keywords:** Software Quality Modelling, Coding Standards, Software Metrics, Statistical Analysis  
ISO 9126 Quality Model

The ISO 9126-1 standard [4] has been introduced to formalise the notion of Quality of a Software System. 3 distinct aspects are considered:

- Internal Quality measured for a non-executable form of the Software System, e.g. its source code.
- External Quality, which pertains to the run-time behaviour of the system, as experienced during dynamic test.
- Quality in use, which addresses the degree to which user goals and requirements are fulfilled.

Internal and External Quality can be further categorised into 6 separate characteristics:

- Functionality

- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Each of these 6 characteristics can be further subdivided, and there are 27 sub-characteristics in total.

Quality in Use has been divided into 4 characteristics:

- Effectiveness
- Productivity
- Safety
- Satisfaction

ISO 9126-1 advocates measuring each of these characteristics, but does not specify how. Examples of suitable metrics are given in Technical Reports: 9126-2 [5], 9126-3 [6], 9126-4 [7]. The standard stipulates that with suitable choices of metrics Internal Quality should predict, or in other words correlate with External Quality, which in turn should predict Quality in Use.

In this study we will be focusing on the Satisfaction Quality in Use characteristic. We will attempt to demonstrate that this characteristic can indeed be predicted by measuring Internal Quality of a software system, see Section 0. We will also be examining empirical evidence of a correlation between Internal and External Quality measures, see Section 0.

Prior to conducting such a study we needed to settle on suitable metrics for Internal Quality. ISO 9126-3 [6] is a Technical Report that proposes such metrics. The vast majority of them are of the following form: percentage of items (functions, variables, etc.) meeting a specific requirement. There are a number of problems with such a definition of metrics. Their calculation cannot be easily automated, and their value needs to be determined by comparing implementation and design documents with specification. These metrics indicate how much work on the project has been completed, rather than the underlying quality of the implementation. Such

metrics represent good project management practice for green-field projects, and cannot be applied easily when part of the system is re-engineered. Lastly, quality or lack thereof is not seen as an attribute of source code, as none of the proposed metrics are based on direct measurements on source code. This is against the guidance of ISO 9126-1 [4] page 15.

Prior to ISO 9126 there has been a vast amount of research devoted to software metrics [2]. These traditional metrics, such as Cyclomatic Complexity or Estimated Static Path Count, are concerned with the structure of a function, vocabulary of a source file, etc. Therefore, they may yield the same values for drastically different versions or stages of a Software Product, e.g. Cyclomatic Complexity for pseudo code stage may be the same as for the final implementation. Moreover, there is no well-defined and substantiated mapping for these metrics to ISO 9126 characteristics. We examine possible correlations of such software metrics with Quality in Use metrics in Section 0.

## **Coding Standards**

Nowadays, increasingly more emphasis is given to following best practice, and defining and enforcing coding standards, especially for high cost of failure software projects. Compliance to a coding standard is often treated as a pass/fail test. However, a different approach is possible, where the level of compliance is measured, either as the absolute number of violations for a particular source file, module or component, or normalised by the size of the entity, e.g. number of lines of code. This would allow correlating compliance with measurements of other aspects of the product, e.g. run time behaviour or user experience.

The most popular coding standard in the public domain for the C language is MISRA-C [12][13]. It constitutes a subset of the C language that restricts usage of poorly defined or unsafe constructs. Less emphasis is given to presentational aspects: naming conventions and layout. Until recently, no such definitive coding standard was available for the C++ language. The first and probably most complete is High Integrity C++ [14]. More recently, the Joint Strike Fighter Air Vehicle C++ Coding Standards [10] were

released, demonstrating the growing industrial acceptance of using coding standards. Other C++ guidelines tend to focus on specific programming aspects [3][11][16][17].

The rules of these coding standards represent common pitfalls with developing in the corresponding programming language, and have been derived either from experience or on theoretical grounds, by examining the language specification [8][9]. Therefore, counting the number of violations of such rules in a Software Product appears well founded, and intuitively corresponds to a measure of its Internal Quality. This proposition is rigorously evaluated in Section 0.

## **Qualitative Results**

We wanted to verify the proposal for measuring Internal Quality of a software product with real-world examples. We have engaged with some software companies, to find out what tangible benefits enforcement of a coding standard has given them. Two of them were able to offer broad qualitative statements, and these are documented in Section 0 and 0. However, they could not provide, in time for publication of this paper, any numerical data that would allow us to compare, for example, faults found in the field and compliance to a coding standard of specific software modules. However, another company had such data available, and we worked together to establish whether there were any correlations, see Section 0 for details.

### **Company A**

They have been using MISRA-C:1998 [12] ever since historical process data have been collected. Some extra rules are enforced to do with naming conventions and limiting undefined behaviour. Typically, approximately 90% of the rules of this combined coding standard are adhered to for a project.

For a number of specific projects porting from one platform to another was required, and this was achieved with hardly any re-coding. This result was attributed to restricting undefined and implementation defined behaviour in their coding standard.



In their development process, unit testing occurs on a parallel track to coding, review and bench testing. By examining process data it was found that all the faults found in unit testing were also identified in the development track during code review (of which coding standard compliance is a part) or bench testing stages. Therefore, unit testing, despite being part of industry best practice, did not yield any new issues, apart from fulfilling its secondary role of verifying the specification. Subsequently, for some projects unit testing has been limited or dropped altogether in preference to proceeding straight to integration/system test.

### **Company B**

The AUTOSAR[1] subset of MISRA-C:1998 [12] is used, as well as other proprietary coding standards, depending on the project, and this is mandated contractually.

The software projects are large, typically around 500KLOC. By defining a software platform, and making it conform to stricter rules on limiting implementation defined behaviour, they were able to migrate from one compiler and micro controller combination to another in a matter of weeks. This result is similar to that of Company A, see section 0.

Reuse is very common across projects, and coding standard rules on layout and naming conventions were found to be helpful in this regard.

### **Quantitative results**

Company C has an ongoing programme for improving customer satisfaction. To this end they are collecting software fault reports from the field, and tracking them on a regular basis. The incidence of critical software faults tends to vary across their products, and the intention is to identify measurements on source code, i.e. Internal Quality metrics, that would correlate with these fault data, i.e. Quality in Use: Satisfaction metric. Once such source code factors are identified, it will be possible to re-engineer the software to

minimise their value; and thus, likely to minimise the incidence of critical faults in released software.

Together with Company C we have collected code metrics for a number of their software products, and correlated them with the corresponding critical fault data. These code metrics fall into two categories:

- incidence of coding standard violations,
- traditional software metrics [2].

The results are documented in Sections 0 and 0 respectively.

### Message Correlation

As a pilot study we focused on 18 software products written in C++, and owned by a single business unit. Critical fault data for each of the products was available, covering a period of 12 months. In order not to disadvantage large projects, we normalised these measurements of Quality in Use: Satisfaction by the size of the corresponding code base, i.e. amount of KLOC.

Rather than narrowing the study to some specific coding standard or guidelines (see Section 0), we decided to include as many coding rules as possible, in our search for the ones that will correlate with the fault data. QA C++, static analyser for C++ from Programming Research, includes nearly 900 rules ranging from ISO Compliance and Undefined Behaviour [9], Best Practice [3][11][14][16][17], to code layout conventions. This includes rules pertaining to individual source files as well as issues occurring across files, see Table 1 for examples.

confid ence	msg#	QA C++ message text
99.5%	1512	'%1s' has external linkage and is declared in more than one file.
99%	1508	The typedef '%1s' is declared in more than one file.
99%	2085	For loop declaration of '%1s' is hiding existing declaration.
99%	4239	Class type control loop variable

		'% 1s' modified in loop block.
97.5%	4217	Variable '% 1s' is not accessed after this initialisation before it is next modified.
97.5%	4237	Class type control variable '% 1s' not declared here.
97.5%	3600	This 'int' literal is an octal number.
95%	1505	The function '% 1s' is only referenced in one translation unit.
95%	4243	Multiple class type loop control variables found: '% 1s'.
95%	4325	Variable '% 1s' is not accessed further.
95%	4004	Continue statement found.
95%	4208	Variable '% 1s' is never used.
0%	2015	This function may be called with default arguments.

**Table 1.** Message correlation with critical fault data for a sample of QA C++ messages

For every software product we calculated the occurrence of each QA C++ message, and normalised the measurements by the size of the product in KLOC. While we could look for correlations between these raw measurements for fault data and message frequencies, this would make an unnecessary assumption that both of these populations of measurements were distributed similarly.

Instead, we decided to use ranks of the measurements only. If we were to order the software products according to fault data frequency, and for a given QA C++ message according to its frequency of occurrence, similarity between these two orderings would imply a positive correlation between the message and fault data. Considering that we are dealing with a large number of products, from statistical standpoint, it is not necessary that these orderings are identical, for there to be a significant correlation. Given that the number of permutations of 18 entities:  $18! = 18 \cdot 17 \cdot \dots \cdot 2 = 6,402,373,705,728,000$  is a staggeringly large number, if a pair of orderings is within the 5% group that are the

most similar, we can say with 95% confidence that they are correlated. 95% confidence interval is usually considered the minimum level to achieve statistical significance.

This leaves the question of how we are going to judge similarity between two given orderings of 18 products. Spearman's Rank Correlation Coefficient  $R_s$  [15] is a non-parametric statistical test, meaning that it works on the ranks of measurements. It evaluates to 1.0 if the orderings are exactly the same and -1.0 if they are exactly opposite, i.e. one is an inversion of the other sequence. The closer the value of  $R_s$  to 0 the less similar both orderings are. In this study we are only interested in positive correlations between Quality in Use and Internal Quality metrics:  $R_s > 0$ . Given that we are dealing with 18 products, in order to have 95% confidence of a positive correlation between QA C++ message and fault data, the value of  $R_s$  needs to be no smaller than 0.401. Table 1 documents critical values of  $R_s$  for higher confidence intervals.

Confidence	95%	97.5%	99%	99.5%	99.9%
Critical Value of $R_s$	0.401	0.472	0.550	0.600	0.692

**Table 2.** Critical Values of Spearman's Rank Correlation Coefficient  $R_s$  for 18 entities

The first 12 rows of Table 1 list QA C++ messages that are positively correlated with critical fault data for the 18 software products under consideration, with at least 95% confidence. As an illustration the last row contains the message that has the value of  $R_s$  closest to 0. Figures 1-5 on page 142 display the correlation between the ranks of fault and message frequencies for each software product as a scatter plot, for a representative selection of messages from Table 1. Dots (software products) that lie on the  $y=x$  (diagonal) line represent complete agreement between the ranks. In Figure 1 dots are much closer to the diagonal line than in Figure 5, which visually confirms the accuracy of the Spearman's Rank Correlation Coefficient. Figure 6 corresponds to the message with the smallest value of  $R_s$ ; for convenience both positive  $y=x$  and negative  $y=19-x$

correlation lines are drawn. As can be seen dots are equally distant from both diagonal lines.

This result can be interpreted as follows: there is at least 95% likelihood that 12 QA C++ messages detailed in Table 1 are positively correlated with critical faults in 18 software products under consideration. This allows us to assume that by re-engineering these products to reduce the incidence of these messages, future occurrence of critical faults may also be reduced. As the organisation is interested in improving customer satisfaction, targeting these messages and monitoring their frequency can supplement the existing quality procedures.

It is worth pointing out that these 12 recommended messages are Best Practice rules, rather than rules targeting Undefined Behaviour, e.g. array access out of bounds, or division by 0. Such rules targeting potential ‘bugs’ are unlikely to occur frequently in the code. If for 18 products most frequencies are 0 apart from a few, the Spearman’s Rank Correlation Coefficient will not exceed the critical value, and so the corresponding QA C++ message will not be flagged up as correlated with critical fault data. Therefore, it is necessary to supplement rules/messages identified by this statistical procedure with rules targeting bugs, portability issues, and other priorities identified for the software products in question.

## **Metrics Correlation**

Apart from looking for correlations between critical faults and QA C++ messages, we were interested in examining whether traditional software metrics [2] could be of use. QA C++ calculates several function, file and class based metrics. We have recorded the average, maximum and standard deviation value of every metric for each of the 18 software products. We then calculated the values of Spearman’s Rank Correlation Coefficient  $R_s$  between the critical fault and these metric data across the 18 products, which are collected in Table 3. Critical value of  $R_s$  at 95% confidence level is 0.401, and none of the metrics meet that for either average measurement, maximum or standard deviation. Therefore, we could not recommend any of these software metrics to be included in the quality initiative.

## Summary

In this paper we have proposed using coding standards compliance as a measure of Internal Quality of a Software System. The validity of this metric has been confirmed on a group of real-world software products, as for a number of coding rules it was found to correlate with a metric for Quality in Use: Satisfaction characteristic. Also, compliance to a coding standard has been found by two separate organisations to positively impact External Quality: Portability characteristic of their software.

User satisfaction is a concrete concept, and can be measured, e.g. by recording faults in released software. Coding standards compliance can also be easily measured, and subsequently improved, but does not directly map to improved user experience. However, this could be inferred, if a correlation between user satisfaction and compliance to coding rules is found, as is the case in this paper.

An interesting topic for a future study would be to empirically demonstrate validity of this cause and effect hypothesis, by examining whether incidence of faults will be reduced in proportion to improvement in coding standards compliance.

Metric	avg	max	Std dev
Class metrics			
Coupling to other classes	0.041	0.005	0.043
Deepest inheritance	0.083	0.166	0.100
Lack of cohesion within class	-0.012	-0.061	-0.046
Number of methods declared in class	-0.098	-0.020	-0.023
Number of immediate children	0.055	0.025	0.061
Number of immediate parents	0.055	0.034	0.055
Response for class	-0.031	-0.057	-0.031
Weighted methods in class	-0.017	-0.034	-0.069
Function metrics			

Cyclomatic complexity	0.087	-0.141	-0.234
Number of GOTO's	-0.153	-0.238	-0.154
Number of code lines	-0.061	-0.068	-0.256
Deepest level of nesting	0.103	0.234	0.087
Number of parameters	0.129	0.192	0.122
Estimated static program paths	-0.362	n/a <sup>2</sup>	0.084
Number of function calls	-0.102	-0.019	-0.239
Number of executable lines	0.017	0.018	0.009
File metrics			
Comment to code ratio	0.283	0.153	0.287
Number of distinct operands	-0.220	-0.239	-0.304
Number of distinct operators	-0.035	0.260	0.124
Total preprocessed code lines	-0.074	0.142	-0.087
Total number of tokens used	-0.144	0.040	-0.138
Total unpreprocessed code lines	-0.073	0.077	-0.117
Total number of variables	-0.187	-0.044	-0.261

**Table 3.** Metrics correlation with fault data

Critical value of Rs at 95% confidence level is 0.401

## References

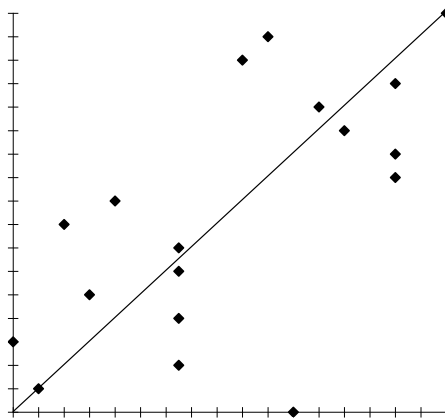
- [1] Automotive Open System Architecture, [www.autosar.org](http://www.autosar.org)
- [2] N.E. Fenton, S.L. Pfleeger. Software Metrics: A Rigorous Approach. 2nd edition. PWS, Boston, 1998
- [3] M. Henricson, E. Nyquist, N. Erik. Industrial Strength C++: Rules and Regulations. Prentice Hall, 1997
- [4] ISO/IEC 9126-1:2001. Software engineering – Product quality – Part 1: Quality model.

---

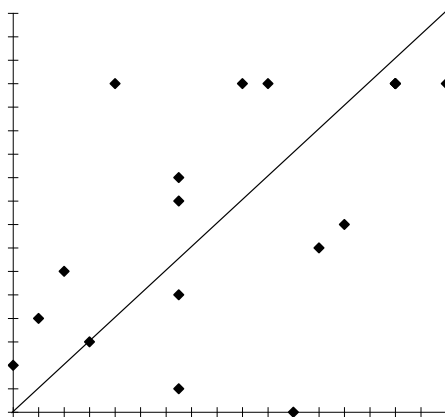
<sup>2</sup> For technical reasons we were not able to accurately calculate this value.

- [5] ISO/IEC TR 9126-2:2003. Software engineering – Product quality – Part 2: External metrics.
- [6] ISO/IEC TR 9126-3:2003. Software engineering – Product quality – Part 3: Internal Metrics.
- [7] ISO/IEC TR 9126-4:2004. Software engineering – Product quality – Part 4: Quality in use metrics.
- [8] ISO/IEC 9899:1990. Programming languages – C.
- [9] ISO/IEC 14882:2003. Programming languages – C++.
- [10] Lockheed Martin Corporation. JSF AV C++ Coding Standards.  
<http://www.research.att.com/~bs/JSF-AV-rules.pdf> 2005
- [11] S. Meyers. Effective C++: 55 Specific Ways to Improve Your Programs and Designs. 2nd edition. Addison Wesley, Boston, 2005
- [12] MIRA, MISRA-C:1998 - Guidelines for the Use of the C Language in Vehicle Based Software. [www.misra-c.com](http://www.misra-c.com), 1998
- [13] MIRA, MISRA-C:2004 - Guidelines for the use of the C language in critical systems. [www.misra-c.com](http://www.misra-c.com), 2004
- [14] Programming Research. High Integrity C++ Coding Standard Manual. [www.codingstandard.com](http://www.codingstandard.com), 2004
- [15] S. Siegel. Nonparametric Statistics for the Behavioral Sciences. McGraw-Hill Book Company, Berkshire, 1956.
- [16] H. Sutter, A. Alexandrescu. C++ Coding Standards: 101 Rules, Guidelines, and Best Practices. Addison Wesley, Boston, 2004
- [17] H. Sutter. Exceptional C++. Addison Wesley, 1999

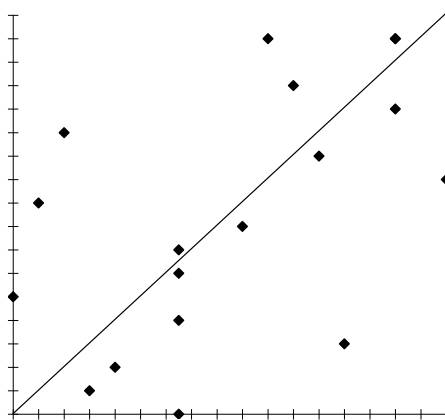




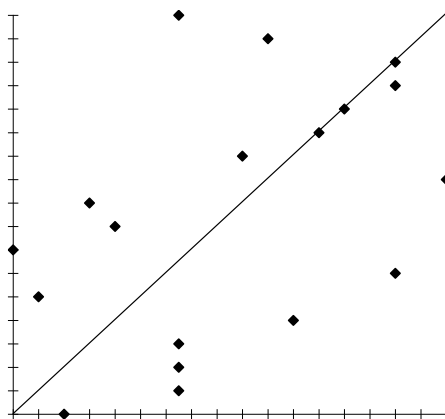
**Figure 1.** correlation for message 1512  
 $R_s=0.649$ , confidence interval 99.5%



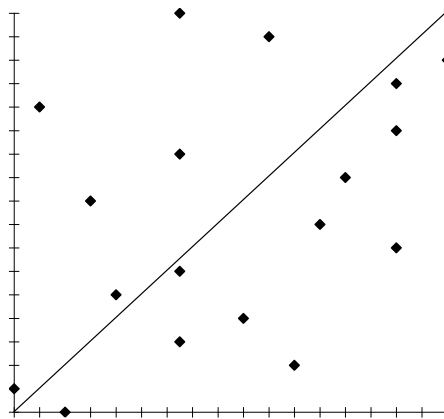
**Figure 2.** correlation for message 1508  
 $R_s=0.568$ , confidence interval 99%



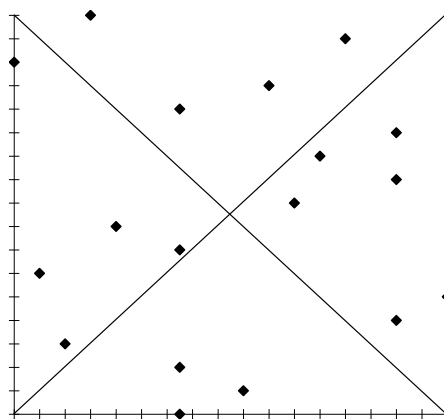
**Figure 3.** correlation for message 4217  
 $R_s=0.533$ , confidence interval 97.5%



**Figure 4.** correlation for message 1505  
 $R_s=0.466$ , confidence interval 95%



**Figure 5.** correlation for message 4208  
 $R_s=0.403$ , confidence interval 95%



**Figure 6.** correlation for message 2015  
 $R_s=0.001$ , i.e. no correlation

# **Internal Training Program for SQA, is it Worth for Money?**

**Yelena Belyayeva**  
**Motorola, St.-Petersburg software center**  
**St.-Petersburg, Russia**  
**email: Yelena.Belyayeva@motorola.com**

## **Abstract**

The paper is devoted to the training program for Software Quality Assurance (SQA) engineer created in Motorola, St.-Petersburg software center to prepare SQA engineers to fulfill their responsibilities. The paper will provide the stages how the training program was formed. We will describe approaches we used for the mentoring program and the effectiveness of the training program.

**Keywords:** Software Quality Assurance; training; education.

## **1. Introduction**

You may ask why an idea to run internal training program for SQA people occurred to us and why we spend up to 6 months to prepare one person when it seems easier and cheaper to hire a specialist with required skills. There are several reasons that will be explained: first, historical, when the center started its work, we were the first who followed the requirements CMM/CMMI model [1] describing quality assurance activities. So we had a limited conception of SQA responsibilities based on our understanding of CMM/CMMI statements and experience of our colleagues from other Motorola centers. The second reason was lack of educational programs related to quality assurance in St.-Petersburg and even in Russia. Now when they are appearing most of them are still mixed with testing concepts. Third, while the center was growing we were obtaining specific experience, e.g. in auditing approaches and organization metrics database maintenance. All these reasons led to the decision to introduce internal training program for SQA newcomers and to start training newcomers by ourselves.

## 2. Training Program Formation

When the center started its work in St.-Petersburg in 1997 nobody knew what requirements for SQA engineer position should be. The primary source was a process area of CMM model (and in the course of the time CMMI model) used in a number of Motorola centers. This process area is devoted to process and product quality assurance. The main requirements are objectively evaluating of performed processes and work products against defined process, reporting of noncompliance issues and ensuring that noncompliance issues are addressed. Looking at CMM model more detailed we will find out additional SQA responsibilities: contribute to project or organization process definition and its improvement, maintain organization metrics database and participate in metrics analysis both at project and organization level. So SQA engineer plays several roles simultaneously: auditor, metrics expert, process and problem prevention consultant.

Since 1997 the organization grew more than 5 times that led to the shortage of SQA personnel to maintain the increasing number of projects/programs. Moreover the diversity of processes, number of process assets and project tailoring scope increased. To keep the system working and up-to-date we introduced automation into the following areas: audit system, organization metrics program and tracking system for process changes. As a result we needed knowledgeable people who can easily manage the innovations and changes in work situation.

First, we looked at the labor-market and found out that it was hard to find a person with such qualification. There was no any special educational program in universities, and there was no consistency in how CMMI world treated the term of Quality Assurance vs. other world of software production. Most of software companies considered and continue considering QA people as testers. This also brought an idea to train and bring up quality people ourselves. We defined the minimal requirements for SQA candidates that were vital to start the work successfully:

- computer science education to be on the same level of language with the projects;
- basic knowledge of databases and metrics tools;
- communication skills as most of time QA engineers had to deal with project teams.

Along with searching process we worked out the training program that would cover both theoretical and practical parts of QA activities.

We tried to look at specially developed courses or certifications for quality assurance, but there were no such courses in Russia and any international certification implied expenses on travel to the place of certification. Additionally due to difference in terminology most of the courses were oriented at testing activities and anyway they gave only general knowledge of the subject: common approaches to auditing, common metrics, etc. However by this time we had a highly-matured organization process with a lot of specific features, even auditing approach was different. So we started with theory and prepared an induction program for newcomers at SQA position. Then we focused on the second part of training program - practice that can be obtained only through work with projects. We didn't want to let newcomers work directly with projects after induction program was completed. What was good in 1999-2003 years did not work in 2004. Our experience showed us the necessity of controlled practice and as a result a mentoring program was introduced.

The created training program contains several parts which are delivered by senior SQA staff:

- Induction training program which provides details in all process and quality areas which are essential to perform SQA duties efficiently. The program was established in 2004 and is being maintained up-to-date
- Mentoring phase with close cooperation of a mentor and a mentee within several selected projects
- Post-induction mentoring phase when a mentee works unassisted and involve the mentor into critical issue resolution

### **3. Mentoring Program**

After completing induction training program a mentee starts his/her work with 4-5 projects under mentors' supervision. The first approach to mentoring was the following: there was one or a couple of mentees and several mentors depending on what projects they worked with. In this case mentees received an experience in most of project processes accepted in different programs. On the other side, there were more minuses then pluses as there was no single person of contact (you could ask any mentor, but he/she wasn't an expert in

the problem of other program), the learning progress was slow and the duration of mentoring phase was unacceptable. After the failure with the first approach we changed it the approach when one or a couple of mentees contact only one mentor. In this case we faced the problem when a mentee had a limited scope of projects where mentor worked. But it was not a real problem as by this time we had introduced program differentiation when SQA engineers had been assigned to the projects from not more than 2 or 3 programs. The gap in knowledge of other program's process was closed during sharing sessions within SQA group. For the mentoring phase we introduced coverage matrix, checkpoints and the rules for mentors and mentees. Coverage matrix is a table of regular project/organization tasks implying SQA involvement (e.g. auditing, reviews of project documents, project meetings, metrics update) and the status of their implementation. The practice was considered as complete when it had been done at least 2-3 times: first one with detailed instructions from the mentor and the second or third one unassisted with showing the results to the mentor. During the checkpoints a mentor checked the status of coverage matrix filling in and level of readiness to work unassisted. Step by step the mentors delegated more and more tasks. With the conception of single point of contact and project scope limitation we improvement the process of learning and shortened the mentoring phase duration to not more than three months.

Three generations of new SQA engineers were covered by the program from 2004 year (8 people). The duration of post-induction mentoring was shortened 3 times (from 1.5 year at the beginning to up 6 months). From mid of 2006 year in spite of the fact that SQA team consisted mostly of fresh SQA engineers the results of internal feedback survey showed maintained level of SQA competence as exceeding expectations.

#### **4. Conclusion**

In conclusion we would like to stress that to train specialists it is not enough to give them only theoretical knowledge but it is essential to obtain practice under mentors' supervision.

We hope you can use our experience to build your own training program for specialists that cannot be found easily at labor-market.

## **5. References**

[1] Capability Maturity Model Integration (CMMISM) for Software Engineering, Version 1.2, CMU/SEI-2006-TR-008, ESC-TR-2006-008, August 2006



# **CMMI and Agile – is convergence possible? An example of international distributed project using combined process**

**Ivan Gumenyuk**  
**EMC Corporation**  
**S.Petersburg Center of Excellence**  
**email: Gumenyuk\_Ivan@emc.com**

## **Abstract**

This article suggests one more consideration of the advantages and drawbacks of two fundamental approaches to software development and the possible combination of elements from both of them to create a software development process for a new, large software project under the conditions of internationally distributed development. The article briefly lists the strengths and weaknesses of each approach, depending on other conditions, the environment and functional area. For example, the method for the formation of the software development process in a project being developed in four geographical locations by American and Russian software engineers; how the transition from the Agile-like style to a combined approach started and evolved, what the reasons were for that, which advantages were obtained and which problems still need to be solved. The article also raises the question about the necessity of looking for new software development methodologies, which is caused by the growing trend of reusing "off-the-shelf" software components to create new systems. Projects with a high percentage of reuse require most of their efforts during the design and integration phases and for overall stability of a system, as opposed to the traditional approach of creating systems "from scratch" in which most of their efforts are spent on coding.

**Keywords:** Agile, CMMI, software development process.

# **CMM и Agile – возможен ли симбиоз? Пример интернационального распределенного проекта с использованием комбинированного процесса**

**Гуменюк Иван Валентинович**  
**Санкт-Петербургский**  
**Центр Разработок EMC**  
**Gumenyuk\_Ivan@emc.com**

## **Тезисы**

В докладе предлагается рассмотрение достоинств и недостатков двух основополагающих подходов к процессу разработки программного обеспечения, и комбинирование элементов обоих из них в ходе становления процесса в новом проекте создания крупного программного продукта в условиях распределенной интернациональной разработки. Дается краткий обзор преимуществ и недостатков каждого из подходов в зависимости от областей применения. Рассказывается о том, как происходило формирование процесса разработки в крупном проекте, выполняемом в четырех географически удаленных офисах американскими и российскими разработчиками – переход от Agile методов к комбинированному подходу, какие факторы потребовали этого, какие преимущества это дало, какие проблемы пока не решены. Также ставится вопрос о необходимости поиска новых методологий разработки программного обеспечения, что вызвано усиливающейся тенденций переиспользования готовых программных компонентов при создании новых систем. При такой разработке основные усилия требуются на этапе интеграции компонент и для обеспечения надежности системы в целом, в отличие от традиционно рассматриваемой практики написания программного обеспечения «с нуля».

**Keywords:** Manuscript preparation; formatting of text.

## **1. Введение**

Почему я решил об этом написать? В последние год-два я неоднократно слышал от разных людей, являющихся приверженцами Agile подхода к разработке ПО, что CMM/CMMI подход устарел, что он ведет к усложнению процесса разработки, излишним трудозатратам и вообще это просто старый плохой метод. И, напротив, Agile обладает массой преимуществ и всегда должен использоваться. Однако мой немалый уже опыт в индустрии разработки ПО с такими

утверждениями не совсем согласуется. Я много лет работал в крупной компании со зрелым процессом, основанным на CMM/CMMI. Компания первой в России была сертифицирована на Уровень 5 CMM, и я участвовал в этом. Последние мои два года в этой компании я работал в проекте, который декларировал использование методик и подхода Agile в разработке, однако это было не всегда логично и последовательно и приносило немало проблем – особенно при существовавших жестких плановых датах. Затем я перешел в другую компанию, в новый проект – разработка нового продукта, новое подразделение, новые люди. И если в самом начале речь шла в основном об Agile методиках, то через некоторое время стало понятно, что необходимы и другие подходы, чтобы справиться с проблемами, возникшими в ходе работы и роста проекта. Так некоторые элементы CMM/CMMI появились и в нашем процессе разработки. Я полагаю, что комбинация тех элементов, которые необходимы для решения проблем, и есть наиболее эффективный процесс, неважно больше в нем CMMI или Agile.

## **2. Про CMMI**

Эта глава должна напомнить о том, что такое CMMI. Немного истории, что такое процессные области и какие из них считаются ключевыми и почему. А также, когда и где лучше использовать процессы, построенные на основе CMM/CMMI и о существовании моделей процесса очень похожих на Agile.

### **2.1. Немного истории**

С аббревиатурой CMM часто ассоциируют понятие waterfall model – «водопадная» модель разработки проекта, которая предполагает жесткое деление всей длительности проекта (определенной заранее на основе предварительных оценок) на фазы – написание требований, дизайн, кодирование, тестирование. При этом изменение уже созданных требований на более поздних фазах не предполагается, и если это всё же происходит, то может привести к значительному изменению сроков всего проекта. Видимо, эта ассоциация и является

причиной критики CMMI, также как и необходимость создания подробной технической документации, что считается одной из отличительных особенностей CMMI, унаследованной от CMM.

Однако это не совсем так... Массу информации о CMM/CMMI можно легко найти и в книгах, и в Интернете (Начать можно с [http://en.wikipedia.org/wiki/Capability\\_Maturity\\_Model](http://en.wikipedia.org/wiki/Capability_Maturity_Model)). Но несколько слов всё же нужно сказать.

CMM (Capability Maturity Model) появилась в 1987 году и вначале задумывалась как инструмент для оценивания возможностей компаний выполнять правительственные контракты на разработку ПО. Модель определяет пять уровней зрелости организации и критерии оценки – что необходимо иметь на каждом уровне. Начальный, Повторяемый, Определенный, Управляемый, Оптимизируемый – так переводятся определения уровней (Chaotic, Repeatable, Defined, Managed, Optimized). Критериями оценки служат Ключевые Процессные Области, для каждой из которых определены Цели, Свойства и Практики. Наличие свойств и практик, необходимых для достижения целей каждого уровня и определяет, достигнут уровень или нет. Строгая модель и критерии вели к необходимости создавать довольно значительное количество документации, чтобы была возможность доказать наличие тех или иных практик и свойств (ведь это было необходимо для успешного оценивания). Можно сказать, что основной целью CMM было придание процессу разработки ПО организованности – переход от хаоса к упорядоченным процессам (уровни 2 и 3) и оптимизация этих процессов (уровень 5). Отсюда и необходимость наличия документации, требований, плана, описания тестов, и так далее. Это также работало и на повышение качества финальных продуктов. А так как возник CMM как инструмент для выбора подрядчиков на государственные заказы, это объясняет требования по созданию документации и относительно жесткого планирования – отсюда и Waterfall model. Вспомним также, что появился CMM в 1989 году...

## **2.2. Что хорошего и плохого в CMM/CMMI**

Для большого количества организаций достаточно иметь третий уровень зрелости – Defined. Это всего лишь означает, что у организации имеется набор определенных и описанных стандартных процедур, которые обеспечивают повторяемость выполнения проектов с заданным (оговоренным) качеством результатов. Уровни 4 и 5 – Managed и Optimized – нацелены на управление уже не проектами, а процессами управления проектами и организации в целом, и их постоянное улучшение.

Так как появление CMM было тесно связано с правительственными (в том числе военными) заказами, то логично предположить, что проектируемые системы были большими и высокое качество было одним из основных требований. Это и определило желание иметь подробную техническую документацию. Отсюда и требования по качеству продуктов, так как правительственные заказы, скорее всего, были в таких критичных областях как энергетика, здравоохранение, оборонная промышленность. И waterfall модель в таких проектах, наверняка с большим сроком выполнения, была вполне применима и оправдана.

CMMI (Capability Maturity Model Integration), появившаяся в 2002 году (версия 1.1) как развитие и улучшение CMM, описывает дополнительные процессные области и является более гибкой моделью, допускающей большее разнообразие подходов к управлению проектами. Например, моделей процесса разработки, кроме водопадной, стало рассматриваться гораздо больше – спиральная, и итерационная, модели, например, уже довольно близки к фазам Agile. Но, по-прежнему, главной идеей CMMI остается предсказуемость и высокое качество выполнения проектов.

Практика CMMI	Разработка Требований, документирование	Управление Требованиями	Управление Конфигурацией	Техническое Решение - документирование	Верификация, Валидация и их документирование	Интеграция Продукта
Сложная система, много частей	++	++	+++	+++	+++	+++
Длинный цикл	++	++	++	++	++	n/a
Высокие требования к качеству	+++	+++	+++	+++	+++	+++
Много сотрудников, распределенная работа	+++	++	+++	+++	++	+++
Требования мало меняются	++	+	+	++	++	+
Простая система	+	+	++	+	+	+
Короткий цикл	+	++	++	+	++	n/a
Низкие требования к качеству	++	++	++	++	++	++
Мало сотрудников, работа в одном месте	+	+	+	++	+	+
Требования изменчивы	++	+++	+	++	++	++

Основными Процессными Областями можно считать Управление Требованиями, Разработка Требований, Планирование Проекта, Отслеживание Проекта, Интегрированное Управление Проектом, Управление Конфигурацией, Управление Рисками, Технические Решения, Интеграция Продукта, Верификация и Валидация.

Так как CMM/CMMI предполагают более строгое и формальное планирование и отслеживание проекта, более полное документирование – иными словами, большие накладные расходы - то эти модели вряд ли применимы к разработкам небольших систем, маленьким проектам с небольшим количеством сотрудников, проектам с быстро меняющимися требованиями, коротким циклом разработки. Но для длительных проектов, нацеленных на разработку больших многокомпонентных систем, выполняемых большим количеством сотрудников, возможно расположенными в разных местах, применение модели CMM/CMMI вполне оправданно. Какие преимущества от использования CMMI

очевидны для подобных проектов? А также каковы недостатки этой модели? Попробуем показать в виде таблицы на примере нескольких процессных областей и практик. Знаки «+», «++» и «+++» показывают важность и полезность применения практик для условий, определенных в левом столбце.

Как видно из таблицы, СММ/СММІ подходит для длинных проектов (программ) и больших организаций, когда создаются сложные многокомпонентные системы с поставками новых версий каждые 9-12 месяцев и относительно большим числом сотрудников, возможно, удаленных друг от друга. Эти факторы определяют необходимость хорошего документирования системы – документация должна служить основой работ, чтобы быть уверенным в том, что различные группы имеют одинаковое понимание требований, дизайна, интерфейсов, тестов. Время, потраченное на написание документации, впоследствии компенсируется быстротой нахождения нужной информации любым человеком, упрощается поддержка системы и внесение изменений без потери качества. В случае распределенной разработки есть зависимости между командами, отвечающими за различные компоненты. Это означает необходимость хорошего планирования и отслеживания состояния проекта, чтобы предотвратить блокирование одних команд другими и тем самым избегать задержек. Необходимость качественного управления конфигурацией крайне важна в условиях распределенной работы, и для сложных многокомпонентных систем. То же самое можно сказать и про интеграцию продукта – для сложных систем интеграция должна быть тщательно спланирована, особенно в условиях распределенной разработки. В каких предметных областях разработки ПО могут существовать подобные условия и проекты? Во многих – это относится к разработкам любых крупных систем, например ПО для телекоммуникационной инфраструктуры, системы контроля больших производственных комплексов, системы обработки и хранения данных, авионики, и т.п.

Всё, сказанное выше не столь важно для несложных систем, разрабатываемых несколькими инженерами в одном помещении. Для таких проектов накладные расходы при использовании СММІ неоправданно высоки. Необязательно

писать детальные требования и подробный дизайн, если вся разработка ведется группой из 5-7 человек, ежедневно обсуждающих все детали работы. «Водопадная» модель вряд ли применима, если продукт делается за 3-4 месяца. Тут как раз применение Agile методик вполне оправданно – для маленьких проектов, стартапов, проектов в условиях быстро изменяющихся требований.

### **3. Про Agile**

О методологиях Agile написано (например, можно начать с [http://en.wikipedia.org/wiki/Agile\\_software\\_development](http://en.wikipedia.org/wiki/Agile_software_development) ) сейчас, кажется, уже больше, чем о CMM/CMMI, хотя это и более молодое явление, появившееся в середине 1990-х. Agile Манифест, считающий каноническим определением принципов Agile, был опубликован в 2001 году. Сейчас под словом Agile понимают группу методологий разработки ПО, использующих итерационную разработку, постоянное сотрудничество, адаптивные процессы управления на протяжении всей жизни проекта. Agile предпочитает выполнение работ малыми итерациями с минимальным планированием, что помогает минимизировать риски и быстрее адаптироваться к изменениям. Agile также предполагает большую ответственность разработчиков и минимизацию проектной документации. Важнейшей характеристикой является постоянная работоспособность продукта, частые промежуточные релизы. Широко известен также принцип «test first», то есть вначале разрабатываются тесты для продукта, которые и являются требованиями к его функциональности. Не стоит, наверное, повторяться здесь, описывая различные практики, используемые в Agile.

### **4. От Agile к CMMI – совместная жизнь**

В организации, где я сейчас работаю, процессы разработки ПО всё ещё находятся на этапе становления. Мне кажется, на этом примере можно хорошо показать преимущества и недостатки обоих подходов и эволюцию от Agile к более “CMMI-like” подходу.



Для разработки нового продукта (на базе двух уже существующих) был создан новый отдел. Вначале сотрудников работало не очень много, все они были расположены в Америке, хотя и в двух офисах в разных штатах. Нелишне упомянуть, что для нового отдела отбирали лучших инженеров. Работа велась с использованием методологии SCRUM:

- Ежедневные короткие митинги в малых группах, затем Scrum Of Scrums;
- Трехмесячные итерации, заканчивающиеся базовыми релизами (base releases);
- Итерация разделена на 3 отрезка по месяцу (sprints), на каждый спринт делается планирование работы исходя из имеющегося списка задач (backlog);
- Частые, если не ежедневные, сохранения разработанного кода в общий репозиторий (trunk – используется система версионного контроля Subversion), еженочное построение продукта и выполнение unit tests.

Однако, примерно через полгода стали нарастать проблемы... Первоначальной задачей было изготовление прототипа, своего рода proof of concept. Это было сделано, и на этом этапе взаимодействия квалифицированных инженеров между собой хватало для понимания текущих задач и их успешного выполнения. Но появлялись новые высокоуровневые требования, архитектура продукта продолжала усложняться, уточнялись требования низкого уровня к функциональности продукта. Эта работа велась в основном проектными архитекторами, и из-за роста численности сотрудников все они уже не могли принимать участие в обсуждениях. По мере добавления требований и усложнения дизайна системы, росло число задач и взаимозависимостей между ними. Добавлялись компоненты и функциональность из продуктов-предшественников. Объем кода рос, стало необходимо задумываться о последовательности интеграции компонентов в trunk, чтобы система все время оставалась в рабочем состоянии. Примерно в это время была создана наша инженерная группа в Петербурге – наиболее удаленная и географически, и из-за другого языка общения, и находящая в другом часовом поясе. Оторванные от

основного коллектива, мы испытывали трудности из-за отсутствия документации, которая подробно описывала бы дизайн системы (а порой и элементы архитектуры) и требования к отдельным features. Из-за увеличения параллельно разрабатываемой функциональности, нередко стали случаи «поломки» trunk после некоторых добавлений кода туда. Была образована группа тестирования, которой требовалась четкая информация о содержащейся в релизах функциональности и предъявляемых к ней требований. Таким образом, стала ясна необходимость следующих действий, чтобы улучшить нашу производительность:

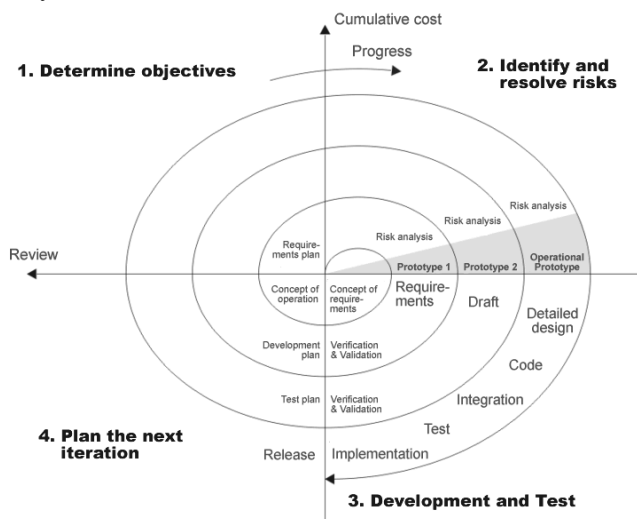
- Создание и обновление системной спецификации, описывающей архитектуру системы, зависимости между компонентами, детализирующую функциональность features и их отображение на модули системы.
- Создание feature teams и их планов работы в соответствии с системной спецификацией (вместо распределения задач из общего бэклога)
- Введение процедур по работе с trunk - набор обязательных действий перед check-in, ограничение check-in to trunk в случае его нестабильности (еще ранее были определены правила работы с ветками (branches))
- Планирование интеграций добавляемых в trunk новых features или их частей
- Создание тестовых планов, общего для продукта и на каждую feature
- Отказ от квартальных релизов и замена их месячными планами, состоящими из набора feature drops, охватывающего интервал в несколько месяцев до момента окончания этапа имплементации
- Упорядочивание процедур работы с дефектами
- Улучшение автоматизированного построения билдов и их тестирования, а также информации о результатах тестирования и состоянии trunk.

Всё это элементы из практик СММІ, они вводились и продолжают добавляться по мере необходимости. Их введение позволило значительно улучшить обмен информацией внутри

отдела, избавиться от простоев в случае нестабильного trunk, улучшить планирование и управляемость проекта. На мой взгляд, это именно пример добавления элементов CMMI в тот слегка хаотичный и Agile-like процесс, который существовал вначале.

## 5. Комбинирование Agile и CMMI

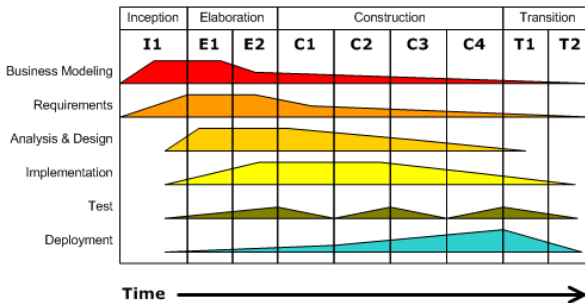
С точки зрения моделирования процесса разработки, существуют модели, совмещающие традиционную «водопадную» модель и итеративные подходы, свойственные Agile – например, спиральная модель, созданная ещё в 1988 году.



Как известно, CMMI может использовать любую модель процесса разработки, а отнюдь не только Waterfall model. Например, такая итеративная модель совершенно не противоречит методологии CMMI:

### Iterative Development

Business value is delivered incrementally in time-boxed cross-discipline iterations.



Совмещение традиционного планирования длительных отрезков времени (например, на полгода) на высоком уровне управления и ежедневных SCRUM митингов в маленьких группах, работающих над конкретными задачами, обычно всегда происходит в крупных компаниях. Правда, при условии, что топ-менеджмент готов скорректировать длительные планы в зависимости от текущей ситуации.

СММІ может обеспечить большую дисциплинированность разработке, ведущейся по методологии Agile. Если количество инженеров, работающих только на trunk, увеличилось и trunk стал нестабильным – значит, надо вводить правила поставки кода в trunk, а это попадает в процессную область СММІ «Управление Конфигурацией», и может быть ещё «Интеграция Продукта». В любом проекте с самого начала существуют управление проектом, планирование и отслеживание – и, если заняться формальным описанием того, как это происходит, то может оказаться что процессы вполне соответствуют СММІ. Так что, неприятие некоторыми горячими сторонниками Agile методологий СММІ может быть лишь свидетельством не очень хорошего понимания этой области.

## 5.1. Добавляем СММІ в Agile...

Когда может быть полезно обратить внимание на «арсенал технологий» из другого лагеря? Если вы работаете в проекте, начатом как Agile, то можно рассмотреть такие ситуации, в качестве примера:

- Увеличилось количество инженеров и trunk стал нестабильным? (подумайте о более формальном подходе к поставке кода в trunk)

- Не получается быстро исправить дефект в версии продукта выпущенной год назад, так как автор кода ушел и никто не понимает как это должно работать? (подумайте о документировании, хотя бы после кодирования, дизайна и интерфейсов)

- Проект разрастается, появился филиал в другом городе? (чтобы избежать потерь времени на поиск информации новыми членами команды, не пора ли написать спецификации системы и разместить их в общедоступном репозитории)

- После окончания тестирования вдруг выяснилось, что одна из функциональных областей не может быть оттестирована? (может быть, было бы полезно заранее дать тестовой группе полные требования, в том числе и к окружению системы)

- Пока вы несколько итераций улучшали и переделывали ваш продукт, выяснилось что конкуренты уже выпустили свой, и ваш уже вряд ли купят? (надо было заранее запланировать даты выхода на рынок, т.е. осуществить долгосрочное планирование)

И так далее... Примеры можно продолжать.

## **5.2. Добавляем Agile в CMMI ...**

Теперь посмотрим с другой стороны, если проект был ориентирован на waterfall-like модель процесса со строгими правилами...

- Написание дизайна затянулось, и сорваны сроки начала кодирования? (Помогло бы планирование начала кодирования сразу после принятия основных решений об архитектуре, вместе с уточнением деталей дизайна, в несколько итераций)

- Группа тестирования говорит что не может начать тестовый цикл, так как разработчики всё ещё не выпустили финальную версию? (Тестировщики могли бы начать тестирование уже давно, отлаживая тесты и попутно находя дефекты начиная с самых первой работоспособной версии. А

ещё лучше, если бы первые тесты были бы готовы ещё до появления первой рабочей версии (принцип test first)).

- Разработчики жалуются, что проведение формальных обзоров кода занимает много времени, а дефекты всё равно потом находят? (сделайте процесс code review более простым, но обеспечьте время для этого своим наиболее опытным инженерам)

- В проекте работает 30 человек и вы больше не можете отслеживать что происходит? (может быть SCRUM в состоянии помочь – проведение scrum-meetings в малых группах и затем scrum of scrums с их лидерами)

Опять же, список примеров можно продолжить...

## 6. Заключение

Итак, на мой взгляд, является очевидным комбинирование методов Agile и CMMI, и если вдуматься, это происходит во многих проектах. Собственно, CMMI просто закрепляет формально многие практики, которые и так используются в развитых и зрелых Agile-проектах. Это мнение не ново – вот что говорит об этом Марк Полк<sup>3</sup>: «...на основании моих методов оценки организация, которая следует только agile-методикам, без труда достигнет третьего уровня модели CMM. В этих методиках нет ничего, что вошло бы в конфликт с требованиями третьего уровня».

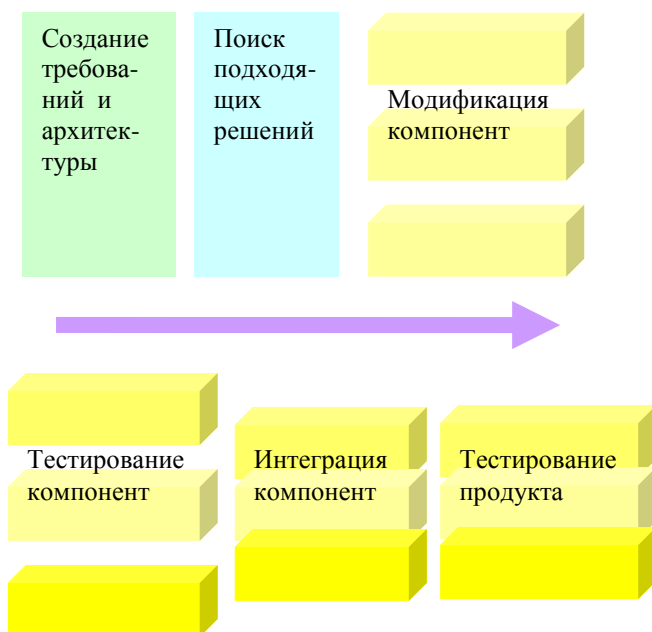
В последнее время всё яснее становится тенденция создания новых продуктов не путем написания кода, а

---

<sup>3</sup> Автор модели зрелости процессов разработки программного обеспечения (Capability Maturity Model, CMM). Марк Полк, в течение полутора десятилетий занимавшийся развитием модели CMM, - признанный гуру в области оценки качества индустриальной разработки программных продуктов. Сегодня он ведет исследования практик высокого уровня зрелости разработки программного обеспечения, а также работает над созданием модели зрелости процессов в ИТ-аутсорсинге eSourcing Capability Model (eSCM).

<http://www.consulting.russee.com/pressroom/inthepress/16-11-2004>

интеграцией уже готовых компонентов, с более или менее значительной модификацией этих компонентов. Это ускоряет вывод продуктов на рынок, но процесс разработки, и модели процесса тут требуются уже иные. Старая waterfall model тут перестает работать совсем, так же как и некоторые методы оценки трудоемкости – ведь фаза кодирования сильно уменьшается. Зато на первое место выходит фаза интеграции и исправления ошибок, а также тестирования. Хорошо, если можно быть уверенным в 100% надежности переиспользуемых компонентов – но чаще это не так; или, сделанные модификации требуют полного тестирования переделанного модуля, причем в составе системы целиком. Процесс создания продукта в таком случае можно изобразить так.



Такой подход к созданию систем может привести к внесению специфических дополнений в CMMI и Agile подходы.

# **Using Reverse Semantic Traceability for Quality Control in Agile MSF-based Projects**

**Konstantin Zhereb<sup>1</sup>, Vladimir Pavlov<sup>1</sup>, Anatoliy Doroshenko<sup>1</sup>,  
and Victor Sergienko<sup>1</sup>**

**<sup>1</sup>International Software & Productivity Engineering Institute  
(INTSPEI)**

**{ zhereb, vlpavlov, doroshenko, sergienko }@intspei.com**

## **Abstract**

Reverse Semantic Traceability (RST) is a quality control method that allows minimizing inconsistencies between inputs and outputs of every step in a software development process. For each step, before proceeding to the subsequent ones, the current inputs are restored (reverse engineered) from the current outputs, and compared to the original versions of inputs. If they are semantically different, then some corrective actions are required to eliminate the inconsistency.

Previously RST was successfully used within projects that employed “formal” methodologies. This paper describes experience of integrating RST into one of agile methodologies (Microsoft Solutions Framework for Agile Software Development). The paper also provides a case-study of using the new combined methodology in a small software development project.

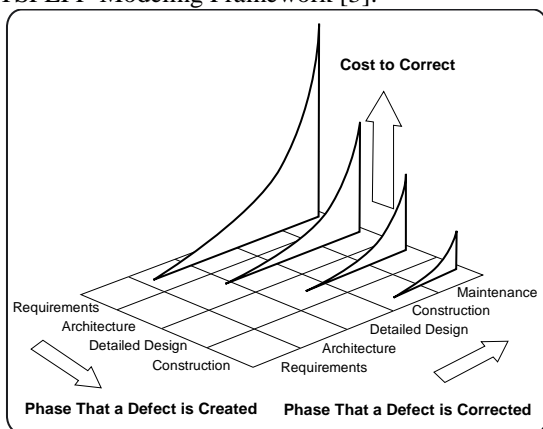
**Keywords:** INTSPEI P-Modeling Framework, agile processes, Microsoft Solutions Framework (MSF), quality control.

## **1. Introduction**

Most currently used Software Development Life Cycles (SDLCs) have the same drawbacks in common. For large projects the most important decisions and the most expensive mistakes are done at the very beginning of the project, and then as the development moves forward the cost of mistakes goes down. This idea is illustrated on Fig. 1 [1]. At the same time, the amount of quality control activities is minimal at the beginning of the project but is increasing as development progresses. Hence, important analysis and design mistakes are usually discovered on the latest phases of the development process which leads to expensive rework.



There are two ways to deal with this problem: to shorten development iterations and make them more frequent, and to incorporate additional quality control techniques to identify analysis and design mistakes when they are introduced – not on the latest phases of development. The first approach became extremely popular recently as agile methods spread over the world [2]. Short iterations allow frequent customer feedback; also automated unit testing can be applied early in the project. However, for large projects the approach of shortening iterations does not work well; there is a need for quality control methods that utilize the second approach. Such methods include various forms of software reviews [9], as well as recently developed Reverse Semantic Traceability – a part of INTSPEI P-Modeling Framework [3].



**Figure 1. Software defects costs**

INTSPEI P-Modeling Framework was first applied in large projects and got successful results described in [5]. Initial feedback from early adopters of INTSPEI P-Modeling Framework suggested that it could be used in agile projects as well. Therefore, we have created an integrated process that combined P-Modeling Framework and one of agile processes – Microsoft Solutions Framework for Agile Software Development (MSF Agile) [16]. Paper [6] describes our experience of integrating P-Modeling Framework with MSF Agile, and also provides technical description of the resulted integrated process. In this paper, we focus on methodological

aspects of applying P-Modeling Framework in agile processes (exemplified by MSF Agile), rather than on technical details. This paper also contains a case study that demonstrates how to use Reverse Semantic Traceability together with agile methodologies.

The rest of the article is organized as follows. In Section 2, we present an overview of Reverse Semantic Traceability and compare it with other similar approaches. Section 3 outlines the integration of P-Modeling Framework with an agile process – MSF Agile. Section 4 describes a small case study of using RST (and P-Modeling) in agile project. Finally, Section 5 summarizes conclusions and presents the directions for future research.

## **2. Reverse Semantic Traceability**

This section provides brief description of a new quality control technique, Reverse Semantic Traceability, which is the most important part of the INTSPEI P-Modeling Framework. Also we compare RST with other similar approaches found in literature.

### **2.1. The idea of the method**

Reverse Semantic Traceability (RST) is a quality control method that allows testing consistency between inputs and outputs of every process step. For each step, before proceeding to the subsequent ones, the current inputs are restored (reverse engineered) from the current outputs, and compared to the original versions of inputs. If they are semantically different, then the step has to be repeated more precisely to eliminate this ambiguous understanding.

The key word in the name of this method is “Semantic” because the original and restored versions of an artifact are to be compared semantically, with a focus on the “meaning” of this artifact, not on particular notions used in it. Hence, the reverse engineering and the evaluation of the semantic difference must be performed by human. Based on this evaluation, a quantitative value can be assigned to each traceability relation between inputs and outputs.

The previous research related to Reverse Semantic Traceability focused on core ideas behind RST [3], as well as its application in

education [4] and in large industrial projects [5]. In particular, paper [5] demonstrated that Reverse Semantic Traceability method could be used successfully for large projects that utilize formalized methods such as RUP or MSF CMMI.

INTSPEI P-Modeling Framework is not discussed here in detail. More detailed discussion of this methodology can be found in [3-8].

## **2.2. Similar techniques**

RST improves existing quality control procedures, such as software inspections [9], by utilizing elements of reverse engineering and traceability management approaches. In this section we compare RST with similar techniques.

RST uses the procedure similar to software reviews and inspections [9-11], when reverse engineers read the “text” of the translated artifact to restore the original artifact. Dunsmore et al. propose the similar technique for reading object-oriented code [10]. However, the distinction of RST is that the reviewer is not allowed to be familiar with the input artifacts – he should restore them from the outputs. RST also adds one more step to the review process, namely comparing restored and original artifact. It helps to ensure that the output artifacts do not conflict previously created input artifacts (e.g. architecture is created according to the requirements or bug fix was performed according to bug description). This additional step also increases the efficiency of the review process, as reviewer (reverse engineer) has to examine the translated artifact more precisely in order to restore the original artifact.

There are also many solutions for capturing traceability relations between project artifacts on different stages of SDLC [12-15]. Traceability is used to establish links between requirements and source code fragments implementing these requirements, as well as to estimate effect of changes in requirements on source code [12]. Research in this area concentrates on metamodels for traceability process [13], usage scenarios of traceability [13, 14], automatic creation of traceability links [15]. However, most traceability solutions provide facilities only for establishing links between artifacts and not for estimating quality of these links. RST approach

enhances traceability solutions by assigning numeric quality values to intermediate traceability links (e.g., from requirements to design).

### **3. Integration with MSF Agile**

INTSPEI P-Modeling Framework is an add-on to existing methodologies, not a standalone process. Currently, INTSPEI P-Modeling Framework integrates both with agile and formal processes. The former include Microsoft Solutions Framework for Agile Software Development [16] and Open Unified Process [17], while the latter include IBM Rational Unified Process [18] and Microsoft Solutions Framework for CMMI Process Improvement [16]. In this paper we focus on integration with agile processes, using the P-Modeling Framework Integrated with MSF Agile as an example. This paper presents only an overview of P-Modeling Framework Integrated with MSF Agile; more detailed technical description is provided in [6].

#### **3.1. Elements of P-Modeling in MSF Agile**

The core elements of the INTSPEI P-Modeling Framework are the Reverse Semantic Traceability and the Speechless Modeling techniques. Both of them were incorporated into the MSF Agile lifecycle. The P-Modeling Framework Integrated with MSF Agile contains detailed descriptions of these techniques and step-by-step instructions for team members. It also describes how Reverse Semantic Traceability and Speechless Modeling work together with other MSF Agile activities.

P-Modeling Framework integrated with MSF Agile suggests performing the following RST activities:

- Perform RST for Scenario;
- Perform RST for Solution Architecture;
- Perform RST for Development Task Implementation;
- Perform RST for Database Task Implementation;
- Perform RST for Bug Fix;
- Perform RST for Scenario Test Cases;
- Perform RST for Quality of Service Requirement Test Cases.

The RST activities are connected to MSF Agile process. When one of the important work products is created according to MSF Agile guidance, P-Modeling Framework recommends performing an RST session to verify that no information was lost or misinterpreted during its creation. The typical RST session consists of the following steps: Preparation, Reverse Engineering Step, Expert Assessment and Making the Decision.

The participants of an RST session assume a set of RST-specific roles, which are active only during a single RST session. P-Modeling Framework adds three new roles to the set of MSF Agile roles: Artifact Owner, Reverse Engineer and Expert, and also uses one of the MSF roles, Project Manager. The RST roles are typically combined with the MSF roles. For example, the person performing the MSF Agile role “Architect” will also perform the RST role “Artifact Owner” during the RST session for the solution architecture. The same person can fulfill different roles in different RST sessions; furthermore, this practice is encouraged in order to increase the understanding of the P-Modeling Framework.

The results of the RST session are captured in specific work products – RST Session Report and RST Expert Assessment. The RST Session Report contains all information about the session, including the date, participants, original and restored artifacts and the final decision. The RST Expert Assessment form captures the result of experts’ meeting – their assessment of quality value and their comments. These work products are used to communicate the results of the RST Session to the team.

Based on the results of the RST session, the Project Manager makes one of the following decisions:

1. The quality of the artifacts is sufficient and the development may proceed to the next phase.
  2. Rework of output artifacts is needed in order to eliminate defects and information loss.
  3. Corrections to both input and output artifacts are needed in order to eliminate misunderstandings
  4. One more RST session after rework of the artifacts is required.
- While the Reverse Semantic Traceability can be applied to all work products, this would create a significant overhead. Therefore, P-

Modeling Framework recommends prioritizing the work products and performing RST only for the most significant of them. The prioritizing is performed by the Project Manager during iteration planning (at the beginning of each iteration). The artifacts are prioritized according to multiple criteria, including their contribution to the quality of the final product, severity of possible defects and availability of other quality control methods. The results of this activity are recorded in the RST Rank Table.

## **4. A case study**

In this section we describe a small case study performed to investigate the usage of P-Modeling Framework in agile project.

### **4.1. Project description**

We applied P-Modeling Framework integrated with MSF Agile in a pilot project. The project consisted in creating an application for simulation of Conway's Life game [19]. The project lasted for 2 months and included 3 part-time student developers.

During the project, the team followed the MSF Agile process, as described in MSF Process Guidance version 4.1 [16]. The project consisted of four 2-week iterations. The first iteration corresponded to MSF Envision and Plan tracks – the team created vision document, collected requirements and outlined initial design. Two intermediate iterations were spent on actual development and testing of the product (Build track). The last iteration combined activities from Build and Stabilize tracks.

The team started using P-Modeling Framework from the beginning of the project. On the first days of the first iteration, the planning of RST activities was performed. Each RST session requires at least one external participant who is not familiar with the project details, and the Project Manager should look for such candidates in advance. Therefore, the Project Manager should understand how many RST sessions are expected on each iteration and for what artifacts.

In our project, the team noticed that RST activities would differ significantly in the first iteration and in all subsequent iterations. During the first iteration, the most important artifacts are vision document and high-level design. Performing Reverse Semantic Traceability sessions to verify these artifacts constitutes a valuable addition to MSF process. Their quality could not be efficiently verified by other means – at least until the implementation is created in the next iterations. In contrast, for all subsequent iterations the focus of RST activities shifts to verifying multiple smaller artifacts: implementations of development tasks, bug fixes and test cases. In these iterations, Reverse Semantic Traceability complements other quality control methods, such as unit testing and functional testing.

P-Modeling Framework integrated with MSF Agile proposes to verify vision statement by restoring it from scenarios (the process is described in “Perform RST for Scenario” activity); the high-level design is verified by restoring the requirements from the design. When the team planned RST activities, they decided that performing RST session for the design will be more valuable. The vision for the project was quite simple, and the team expected that it should not contain any defects. The design, on the other hand, could contain a number of defects, because the student who created it had little experience in design; also miscommunications were possible, as requirements and design were created by different team members. Also the team decided that the importance of design was greater than that of vision.

This paper concentrates on Reverse Semantic Traceability activities that were performed in the first iteration only.

#### **4.2. RST session for Design**

The most important of RST activities for the project was RST session for high-level design. The team created the design of the application in the first iteration. During RST session, reverse engineers were assigned to restore the requirements (in form of scenarios) from the design. The reverse engineers were not familiar with the project before the RST session. When the requirements

were restored, a team of experts compared the restored and original versions of the requirements and expressed their comments.

The initial requirements were split into functional and non-functional (called scenarios and quality of service requirements in MSF Agile). The requirements were stored in Microsoft Team Foundation Server. Fig. 2 shows the scenarios identified by the Analyst. The main scenarios are: editing configuration, running simulation for one or more turns and saving or loading game configuration. Additional lower-priority scenarios included moving through simulation history, changing the size of the displayed configuration and exporting configurations as images. (The full text of the requirements is not included because of space limits.)

The screenshot displays the Microsoft Team Foundation Server interface. At the top, a tab labeled 'All Scenarios [Results]' is active. Below it, a yellow banner indicates 'Query Results: 6 results found (1 currently selected)'. A table lists six scenarios, with the third one, 'Change display zoom' (ID 277), highlighted in blue. The table columns are ID, Scenario, Work Item..., Title, Roug..., Description, and State. Below the table, a yellow banner shows 'Scenario 277 : Change display zoom'. The main area contains a form for this scenario. The 'Title' field is 'Change display zoom'. Under 'Classification', 'Area' is 'ConwaysLife' and 'Iteration' is 'ConwaysLife\Iteration 0'. Under 'Status', 'Assigned to' is 'Semen Nikolenko', 'State' is 'Active', 'Rank' is empty, and 'Reason' is 'New'. At the bottom, there are tabs for 'Description', 'History', 'Links', 'File Attachments', 'Details', and 'P-Modeling'. The 'Description' tab is selected, showing the text: 'In application GUI, be able to zoom in different ways, from 4x4 cells per screen to high-scale views like 10000x10000.'

ID	Scenario	Work Item...	Title	Roug...	Description	State
273	Scenario	Edit configuration		1	User edits colony configuration	Active
274	Scenario	Run turn(s)		1	User executes several Life turns on co...	Active
275	Scenario	Save/load configuration		1	Save colony configuration to file or loa...	Active
276	Scenario	Rewind history		1	Rewind colony history back (and forth...	Active
277	Scenario	Change display zoom		1	In application GUI, be able to zoom in ...	Active
278	Scenario	Export Configuration(s) as (animated) image		1	User saves current configuration or its...	Active

Scenario 277 : Change display zoom

Title: Change display zoom

Classification

Area: ConwaysLife

Iteration: ConwaysLife\Iteration 0

Status

Assigned to: Semen Nikolenko State: Active

Rank: Reason: New

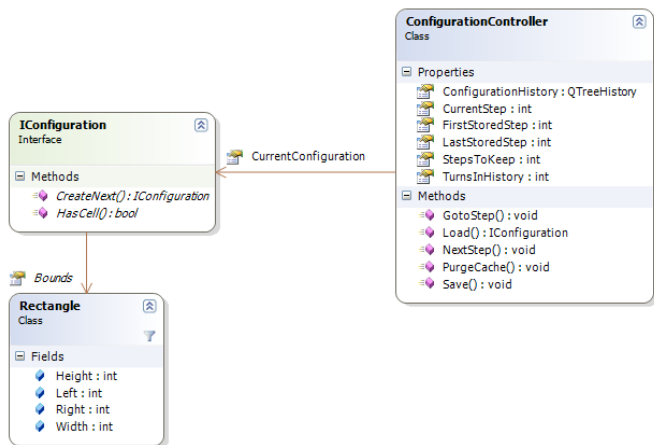
Description History Links File Attachments Details P-Modeling

Description:

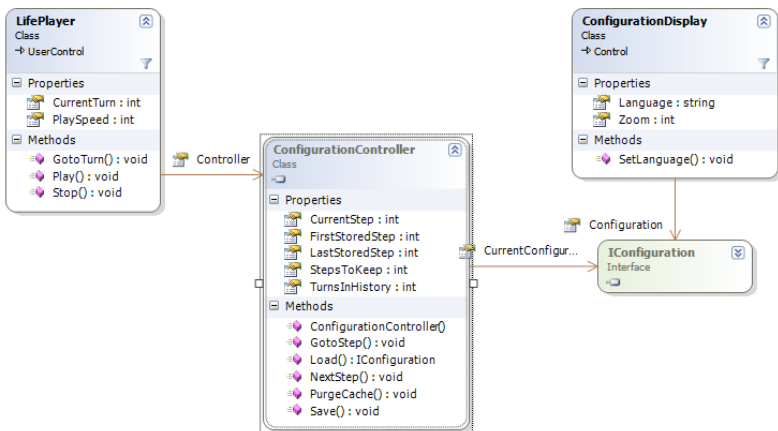
In application GUI, be able to zoom in different ways, from 4x4 cells per screen to high-scale views like 10000x10000.

**Figure 2. Original scenarios**

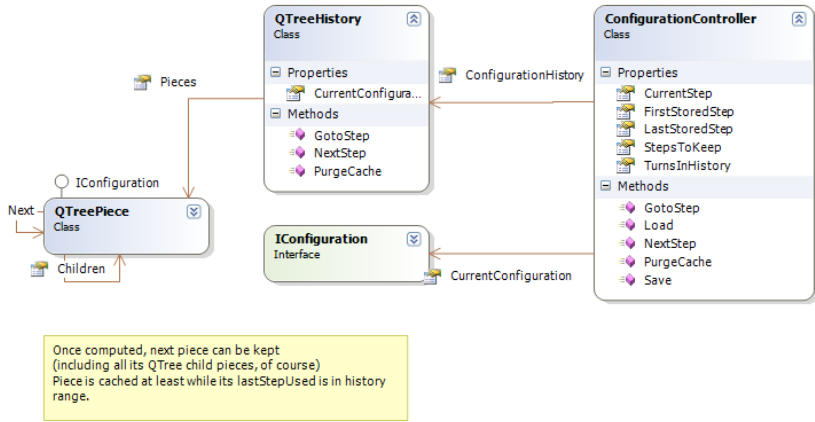




**Figure 3. Core classes**



**Figure 4. GUI classes**



**Figure 5. “Hashlife” algorithm classes**

Based on the requirements found, the Architect developed the application design using Microsoft Visual Studio. The core classes are outlined on Fig. 3. The IConfiguration interface represents single colony configuration; ConfigurationController calculates next game step, stores the history and saves or loads a game state. The classes responsible for graphical user interface (ConfigurationDisplay and LifePlayer) are shown on Fig. 4. Fig. 5 presents classes related to “Hashlife” algorithm [19] that performs simulation.

When the diagrams representing design were ready, an RST session was performed to verify that the architecture meets the requirements. Reverse engineers were provided with the design and were asked to restore the requirements. The reverse engineers had no prior knowledge of any project-related activities; however, they did know the rules of the Life game. The duration of the RST session was restricted to 1 hour. The reverse engineers restored the following requirements (Fig. 6).

After the reverse engineers have restored the requirements, experts compared them with the original requirements. The process took about an hour: first 40 minutes were spent on reading documents by individual experts, followed by 20-minute discussion. The experts produced the following comments (Fig. 7).

Based on the results of the RST session, the team made the following rework decisions:

- Modify requirements – add missing scenario (Choose language)
- Clear up domain dictionary – use “Colony” instead of ambiguous “Configuration”
- Put more work into “Edit configuration” functionality– notable design changes;
- More detailed GUI classes design – minor design changes;

**Scenario 1:** User can control the game of “Life” by performing the following actions:

- Start the game using “Play” command
- Stop the game using “Stop” command
- Move to the given turn using “Go to Turn” command

**Additional Requirements**

- The system simulates each game step with the speed specified by “PlaySpeed:int” parameter
- The systems stores the history of game steps
- Hashlife implementation is used (<http://en.wikipedia.org/wiki/Hashlife>). The fast storage of simulation history is implied.
- There is an option to clear the game history.

**Scenario 2:** The player can change game display configuration. The player can set the display language.

**Additional Requirements**

- Display settings: language, zoom

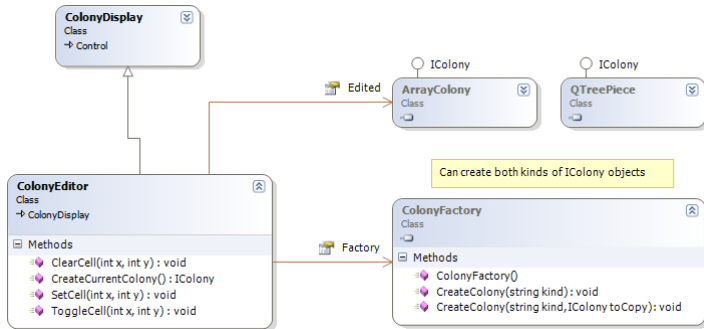
**Notes:**

- User interface provides no possibility to change zoom level and simulation speed
- There is a mistake in Rectangle class: it should contain height, width, left, top, instead of height, width, left, right.

**Figure 6. Restored requirements**

1. Requirement not restored: Edit configuration;
2. Requirement not restored: Save/load configuration;
3. Requirement not restored: Rewind history;
4. Requirement not restored: Export Configuration(s) as animated image;
5. Extra requirement restored: Change language
6. Extra requirement restored: Clear history
7. User interface provides no possibility to change language
8. Note: What is “configuration”? Looks like “program configuration” and can be easily confused with it. I suggest naming it like “colony” or “board”.

**Figure 7. Expert comments**



**Figure 8. Changes in design after RST: a new diagram**

Fig. 8 shows the new class diagram that was created as a part of improvements suggested by the RST session. It contains the classes related to the scenario that was initially omitted from design and hence not restored by reverse engineers (“Edit configuration”). Note that other diagrams have also changed; we don’t include updated versions because of space limit.

### 4.3. Analysis

The RST session performed as a part of MSF Agile project demonstrated that Reverse Semantic Traceability can be a valuable

quality control technique in a project based on agile methodologies. The most important usage scenario for RST is quality control at the beginning of the project, in the first iterations. Even in agile processes, there is a period of time at the project start when no production code is developed. The team focuses on establishing project vision and important architectural decisions. The typical quality control methods of agile processes – automated unit tests and customer feedback – are of limited value in the first iteration. However, the Reverse Semantic Traceability can verify that the first crucial artifacts are correct (or at least consistent).

The team members were excited to discover the method of discovering defects in project artifacts without actually testing the software (or even writing the code). The only other method they knew that could produce the comparable results was software review process. However, the feedback from the team shows that the RST method is more effective. One of the reverse engineers said, “I always thought that design review is a boring process. But the RST session was anything but boring. It was more like solving an exciting puzzle – trying to understand the reasons behind the design decisions ... I believe that RST is more effective than traditional design review. It was the task to restore the unknown requirements that forced us to understand the design better and so find some subtle defects that would evade our attention during design review.”

The RST session proved quite effective in terms of the resources spent and the outcome. The whole RST session required about 7 man-hours: 1 for preparations, 1 hour for two reverse engineers and 1 hour for 4 experts (most of this time was spent by people not directly involved in the project – reverse engineers and experts). However, the outcome of the session was quite significant. Both the requirements and the design have been improved, and some defects were eliminated that would require significant rework were they discovered during actual coding. For example, the “Edit colony (configuration)” scenario that was omitted from design required significant changes in user interface, as well as in core classes. When the improvements suggested by RST were made, the team noticed that one of important assumptions about core classes was wrong. Namely, the QTreePiece class (see Fig. 5) was assumed to be immutable, because of requirements of the “Hashlife” algorithm. However, the “Edit colony” scenario required that the colonies

(implemented by QTreePiece class) could be modified. As a result, the team decided to add one more class representing a colony (ArrayColony) that was mutable. This change in design required about 2 hours. However, the potential cost of rework would be much greater had the wrong design been implemented in code. The team estimated that this defect (missing edit capability) would have been certainly found by the team; but its correction would require significant changes in user interface, core classes and unit tests. The rework would stop the progress of the entire team for a few working days.

The feedback from the case study participants (both project team and RST participants) suggested that Reverse Semantic Traceability effects were not limited to improving the quality of the artifacts that were used in RST session. The team members learned that the project artifacts are not created just because the process says to do so. The artifacts created by one team member will be used by another one, and the author should make the artifact understandable to its subsequent user. The person who created the design said, “I tried to create the design that would be correct and elegant, but I actually forget that it should be understandable. The reverse engineers said they had some problems understanding my diagrams; but if we did not perform RST, the same problems would hinder the progress of developers”. Therefore, RST session ensured that artifacts that were created could be actually used in project. This corresponds to the agile principles stating that the artifacts that are not useful to the project should be avoided. RST helped to make project artifacts, including requirements and design, actually used in project, not just “write-only” bureaucratic burden.

## **5. Conclusions**

We have described an application of Reverse Semantic Traceability technique for quality control in agile project and illustrated it by a case study. The case study has demonstrated that Reverse Semantic Traceability can be used in agile projects, and it provides a valuable addition to agile quality control methods. According to our observations, the usage of RST is the most important during the first iteration of agile project.

The future directions of research include applying P-Modeling Framework in industry projects based on agile process. Also we want to conduct more experiments applying Reverse Semantic Traceability to different kinds of artifacts in order to improve the RST process and create more detailed instructions on usage of P-Modeling in both agile and more formal processes.

## 6. References

- [1] Steve McConnell: Upstream Decisions, Downstream Costs. Windows Tech Journal (1997)  
<http://www.stevemcconnell.com/articles/art08.htm>
- [2] Manifesto for Agile Software Development  
<http://www.agilemanifesto.org>
- [3] Vladimir L Pavlov, Stanislav Busygin, Nikita Boyko, Alexander Babich, "Is There Still a Room For Programmers' Productivity Improvement?", Proceedings of the 5th East-West Design and Test Symposium (EWDTS'07), 2007, pp. 146-151
- [4] Pavlov, Vladimir; Boyko, Nikita; Babich, Alexander; Kuchaiev, Olexii; Busygin, Stanislav., Applying Pantomime and Reverse Engineering Techniques in Software Engineering Education, Proc. 37th ASEE/IEEE Frontiers in Education Conference, Oct. 10 – 13, 2007, Milwaukee, WI. IEEE, 2007, pp. TIE1-TIE5.
- [5] Pavlov, Vladimir; Boyko, Nikita; Babich, Alexander; "First Experience of Using INTSPEI P Modeling Framework in Software Development Projects", Problems in Programming, Issue 2, May 2007, pp. 68-75.
- [6] V. Pavlov, A. Doroshenko, T. Taganskaya, K. Zhreb, N Boyko, An Experience of Integrating INTSPEI P-Modeling Framework with Microsoft Solutions Framework for Agile Software Development, 2007 (accepted for publication in Proc. IASTED Int. Conf. Software Engineering, 2008).
- [7] Pavlov, Vladimir; Yatsenko, Anton "The Babel Experiment: An Advanced Pantomime-based Training in OOA&OOD with UML", 36th 'ACM Technical Symposium on Computer Science Education', February 25, 2005.
- [8] INTSPEI P-Modeling Framework Whitepaper, INTSPEI,  
<http://www.intspei.com>

- [9] Fagan, M.E., Design and Code Inspections to Reduce Errors in Program Development. IBM Syst. J., Vol. 15, No. 3, 1976, pp. 181-211.
- [10] A. Dunsmore, M. Roper, and M. Wood, "Systematic Object-Oriented Inspection—An Empirical Study," Proc. 23rd Int'l Conf. Software Eng. (ICSE '01), pp. 135-144, May 2001.
- [11] Wood, M., Roper, M., Brooks, A., and Miller, J. Comparing and combining software defect detection techniques: a replicated empirical study. SIGSOFT Softw. Eng. Notes 22, 6 (Nov. 1997), pp. 262-277.
- [12] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, Y. Shaham-Gafni, Model traceability. IBM SYSTEMS JOURNAL. Vol. 45, No. 3, 2006.
- [13] B. Ramesh and M. Jarke, "Toward Reference Models for Requirements Traceability," IEEE Transactions on Software Engineering 27, No. 1, 58–93 (January 2001).
- [14] O. C. Z. Gotel and A. C. W. Finkelstein, "An Analysis of the Requirements Traceability Problem," Proceedings of the First International Conference on Requirements Engineering, Utrecht, The Netherlands (1994), pp. 94–101.
- [15] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merio. Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering, 28(10), October 2002.
- [16] Microsoft Solutions Framework  
<http://www.microsoft.com/msf>
- [17] Kroll, Per; MacIsaac, Bruce; Agility and Discipline Made Easy - Practices from OpenUP and RUP, Addison-Wesley Professional, 2006, 448 p.
- [18] Kruchten, Philip: The Rational Unified Process: An Introduction. Addison-Wesley, 2003.
- [19] Conway's Life Game,  
[http://en.wikipedia.org/wiki/Conway's\\_Game\\_of\\_Life](http://en.wikipedia.org/wiki/Conway's_Game_of_Life)



# Intel® Integrated Performance Primitives 2008

**Boris Sabanin**

**Intel**

**boris.sabanin@intel.com**

## **Abstract**

The software product Intel® Integrated Performance Primitives (IPP) and new features are described. The primitives are the building blocks that application developers can easily integrate into their products - applications, components like media plug-ins or high level libraries, in order to significantly increase their performance on Intel and compatible architectures with Windows, Linux or MacOSX operating system installed. The IPP library covers many functional domains: image and signal processing; image coding and data compression; data integrity and cryptography; speech, audio and video coding, and others. Besides of the libraries IPP product contains 50 IPP based Samples released in the source codes. Some of the samples, for example, JPEG2000 image codec and H264 video codec are competitive to the commercial products.

Several important new features were added to the product in 2008 in version IPP 6.0. We will consider two of them in more details - a Deferred Mode Image Processing (DMIP) framework exploiting the CPU cache and multi-core capability, and the IPP functions generated and optimized automatically by a special tool Spiral developed at Carnegie Mellon University.

**Keywords:** Software library, high performance, image processing, signal processing, video coding, image coding, data compressing, automatic code generation, multi-core.

## **1. Introduction**

In spite of the fact that the first signal processing library NSP developed by the IPP team in 1994 failed because Intel and Microsoft could not agree on library's place at the hardware/software territory [1] the team continued design, development and optimization of Intel Performance libraries - Signal Processing SPL, Image Processing IPL, and Speech Recognition RPL. Later, in 2000 the Integrated Performance

Primitives were introduced to improve API of the previous generation libraries. Today, IPP library [2,3] covers more functional domains, works in many operating systems; it is optimized for all Intel CPUs and platforms. The IPP Samples released with the library also have been transformed. At the beginning the samples were just the source codes demonstrating how to use and call IPP. Currently, the samples include quite sophisticated applications such as the Face Detection and Ray Tracing demo applications, and image and video codecs which are strongly competitive to commercial products, for example JPEG2000 and H264 codecs. Designing and developing IPP the team introduces new technologies and applies new tools to improve the development and optimization process as, for example, it started to do with the Spiral tool [4]. We will present the first results of the automatically generating library codes in the paper.

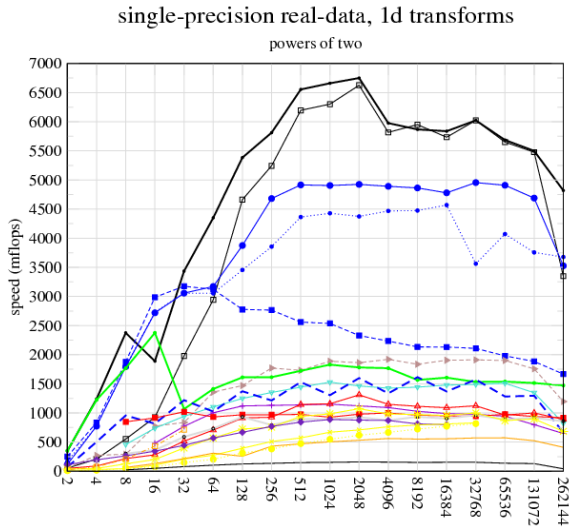
## **2. IPP functionality**

IPP is a library containing 10K functions. Many of them are optimized for IA32, Intel®64, IA64, and Atom™. The library works in operating systems Windows, Linux, MacOSX and QNX. IPP consists of 16 functional domains that cover Signal and Image processing, Speech, Audio and Video coding, String processing, Computer Vision, Speech Recognition, Jpeg & Jpeg2000, Lossless Data Compression, Cryptography, Realistic Rendering, Data Integrity, Vector Math and Small Matrix operations. Additionally to the library IPP customers get more than 50 IPP Samples – applications and components released in source codes. They are video codecs MPEG2, MPEG4, H264, VC1 and AVS; audio codecs MP3, AAC, AC3; Jpeg and Jpeg2000 codecs, speech codecs G722, G723, G726, G728; Face detection demo application, Deferred Mode Image Processing framework, Ray Tracing viewer, Data Compression libraries and utilities GZIP, ZLIB, LZO, BZIP2; interfaces for Java, C#, VB, F90, and C++.

One of the sources where the ideas and new functionality requests come from is IPP forum at Intel Software Network web site [5]. The IPP team is actively involved into discussions with IPP users and is very proud of the activity level the IPP customers ask questions, complain of issues and propose solutions.

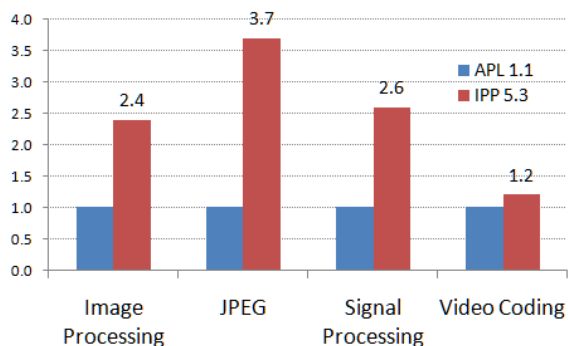
### 3. Performance provided in IPP

One of the performance indicators characterizing any signal processing library is performance of the FFT/DFT transform a library provides. According to FFTW [6] in most of the benchmarks FFTW evaluates and presents the IPP FFT is faster than all known FFT implementations. Performance of several single precision read data FFT implementations in MFlops, higher is better, is presented on Fig.1; IPP FFT is on the top.



**Figure 1.** Performance in MFlops of different FFT implementations published on FFTW [3] site.

IPP is fastest in image processing (compared with Matrox), data compression (compared with gzip, zlib, LZO), image coding (compared with IJG), cryptography (compared with OpenSSL). And, finally IPP is a faster library compared to AMD APL library. For example, relative performance of the AMD APL 1.1 and Intel IPP 5.3 functions measured on AMD Opteron system, higher is better (for IPP), is presented on Fig. 2.



**Figure 2.** Relative performance of AMD APL 1.1 and Intel IPP 5.3 functions measured on Opteron.

#### 4. New features in IPP 2008

Several new features have been introduced in IPP 6.0. Shortly, they are: new domain Data Integrity with Reed Solomon coding functions, optimization for i7 CPU (codename Nehalem) and Atom™; the high level data compression libraries; the deferred mode image processing framework, and the signal processing functions generated automatically. Let us consider several of the features.

##### 4.1. Deferred Mode Image Processing

DMIP [7] was introduced in response to a requirement from a strategic IPP customer involved in large-scale image processing. The DMIP framework effectively handles processing of large image data that don't fit entirely within the processor L2 cache. Three main features of DMIP improve image processing capability: a) processing of images by the fragments that fit L2 cache; b) using the highly optimized IPP in such fragments processing; c) parallel processing which could be processing of different fragments or execution of different independent branches of a graph. To start such processing you create a task description as a DAG (directed acyclic graph), and translate the graph into a sequence of IPP calls. To complete (deferred) processing you run the generated sequence

of IPP calls. For example, you need to filter an image by a harmonization filter. The operation can be expressed as

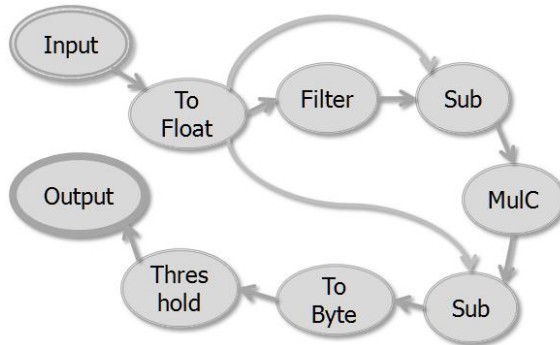
$$D = \min(Tmx, \max(Tmn, (A - (A - Fb(A)) * c))),$$

where Fb is a filter box, Tmn and Tmx are the min and max threshold levels, and C is a constant.

Then at the symbolic level you write C++ code

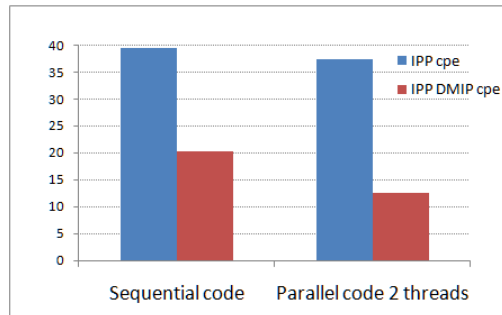
```
Image A, D;
Kernel K;
Ipp32f C;
Ipp8u Tmn, Tmx;
Graph O=To32f(A);
D=Max(Min(To8u(O-(O-O*K)*C),Tmx),Tmn);
```

The DAG corresponding to the code is presented on Fig. 3.



**Figure 3.** DMIP graph for a harmonization filter

The graph can be compiled once and executed many times. Every operation (a node of the graph) operates upon an image slice (DMIP feature a); in most of the cases it is an IPP call (feature b); the slices are processed in parallel (feature c). Processing with DMIP compared with traditional IPP processing could be up to 3 times faster. On Fig. 4, we can compare performance results for the filtering operation on image 2048x2048 with Harmonization filter 7x7 obtained with DMIP and traditional IPP calls.

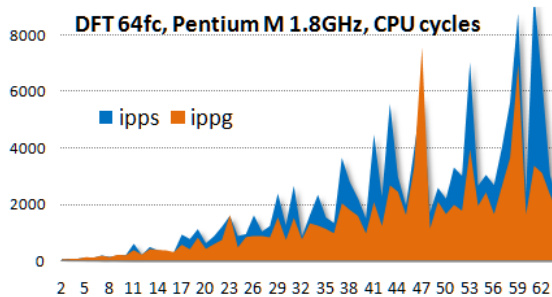


**Figure 4.** Performance of the Harmonization filter operation with IPP and DMIP. CPU cycles per pixel, the less the better.

#### **4.2. Automatically generated IPP functions**

IPP 6.0 includes a new functional domain called IPP Gen. In contrast to other parts of IPP, this library is not implemented by human developers but is entirely computer generated - probably "a first" in high performance library development anywhere. The tool is called Spiral and is developed at Carnegie Mellon University [8].

For given transform to be implemented Spiral generates and evaluates many different possible algorithms represented in an internal math language. Further, Spiral performs optimization such as memory hierarchy optimization, vectorization (for example with SSE3), and parallelization for multiple cores by rewriting mathematical expression. In the end, Spiral outputs the fastest code found which is often faster than existing human written code. On Fig. 5 we compare performance of the Spiral generated code for DFT transform of different size to the IPP existing code in the signal processing library. Performance is given in CPU cycles, the less the better.



**Figure 5.** DFT for complex data of double type. Performance of Spiral generated code compared to IPP existing code. CPU cycles, the less the better.

## 5. Conclusion

Several new important features have been added in IPP 6.0: the deferred mode image processing framework and the automatically generated functions. Both define a direction of the library development: introducing a high abstraction level to better utilize new Intel architecture capabilities and development and optimization automation. IPP is a library which is unique in performance provided on Intel and compatible platforms, in functionality covered, in the technologies the team uses in its development, and in supporting software community.

## 6. References

- [1] Robert A. Burgelman, Strategy Is Destiny, The Free Press, pp. 236, 2002.
- [2] Intel® Integrated Performance Primitives (IPP)  
<http://www3.intel.com/cd/software/products/asmo-na/eng/perflib/219780.htm>
- [3] Stewart Taylor, Optimizing Applications for Multi-Core Processors, Using the Intel® Integrated Performance Primitives, Intel Press, Second Edition, 2007
- [4] <http://www.spiral.net>

[5] Intel® Software Network web site  
<http://softwarecommunity.intel.com/isn/Community/en-US/forums/1274/ShowForum.aspx>

[6] <http://www.fftw.org/speed/>

[7] Alexander Kibkalo, Michael Lotkov, Ignat Rogozhkin, Alexander Turovets, Deferred Image Processing in Intel® IPP Library, Proceedings of the Computer Graphics and Imaging conference, Innsbruck Austria, 2008.

[8] Markus Puschel, Jose M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson and Nicholas Rizzolo, SPIRAL: Code Generation for DSP Transforms, Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation," Vol. 93, No. 2, pp. 232-275, 2005



# **Improved role-based access control model**

**Andrew Mayorov**  
**BYTE-force**  
**email: xor@byte-force.com**

## **Abstract**

Paper covers general principles of building of access-control subsystems in applications and briefly describes base models which are usually employed thereto: mandatory, discretionary and role-based access controls. Paper considers restrictions of base role-based access control (RBAC) model which is widely adopted in modern applications due to relative simplicity of administration.

Paper in details covers features and components of access control model developed by authors, and shows that this model is free from restrictions of base role-based model. It proves that using new model it's possible to implement base discretionary and role-based models. There are also possibilities to combine presented model with mandatory access control.

**Keywords:** access control models, role-based access control.

# **Улучшенная ролевая модель управления доступом**

**А.В.Майоров**  
**BYTE-force**  
**email: xor@byte-force.com**

## **Тезисы**

В статье освещаются общие принципы построения систем безопасности, и дается описание базовых моделей, обычно используемых для этой цели. Детально рассматриваются ограничения ролевой модели, которая, из-за сравнительной простоты администрирования, является предпочтительной к использованию в прикладных программах.

Подробно описываются возможности и составные части разработанной нами модели, и показывается, что она лишена вышеупомянутых ограничений. Доказывается, что в рамках предлагаемой модели можно реализовать базовые ролевую и дискреционную модели, а также обсуждаются возможные пути соединения ее с мандатной моделью. В заключении подчеркивается основное отличительное свойство нашей модели и кратко говорится о преимуществах, которые оно несет.

**Keywords:** access control models, role-based access control.

## **Постановка задачи и результат**

Перед нами стояла задача разработать модель управления доступом к объектам приложения, позволяющую реализовывать любую разумную политику безопасности приложения, оставаясь при этом максимально удобной в администрировании.

В результате получена модель, которая, с одной стороны, совместима с известными базовыми моделями, а с другой стороны, вводит дополнительную степень свободы – привязку пользовательских полномочий к иерархии объектов системы. Подобная привязка позволяет ограничить область действия выданных пользователю полномочий и, как следствие, упростить схемы доступа к объектам.

## **Базовые модели**

Общим подходом для всех моделей управления доступом является разделение множества сущностей, составляющих систему, на множества объектов и субъектов. При этом определения понятий «объект» и «субъект» могут существенно различаться. Мы будем подразумевать, что объекты являются некоторыми контейнерами с информацией, а субъекты – пользователи, которые выполняют различные операции над этими объектами.

Можно выделить три основные модели управления доступом к объектам: мандатную, дискреционную и ролевую.

### **Мандатная модель**

Классической мандатной моделью считается модель Белла-ЛаПадулы [[1]]. Она базируется на правилах секретного документооборота, использующегося в правительственных учреждениях. В этой модели каждому объекту и субъекту (пользователю) системы назначается свой уровень допуска. Все возможные уровни допуска системы четко определены и упорядочены по возрастанию секретности. Действуют два основных правила:

1. Пользователь может читать только объекты с уровнем допуска не выше его собственного.
2. Пользователь может изменять только те объекты, уровень допуска которых не ниже его собственного.

Цель первого правила очевидна. Смысл второго в том, чтобы воспрепятствовать пользователю с высоким уровнем доступа случайно раскрыть какие-то известные ему тайны.

Одной из проблем этой модели считается беспрепятственность обмена информацией между пользователями одного уровня, так как эти пользователи могут выполнять в организации разные функции, и то, что имеет право делать пользователь А, может быть запрещено для Б. Поэтому в практике мандатную модель обычно используют совместно с какой-нибудь другой [[2]] [[3]].

### **Дискреционная модель**

В дискреционной модели безопасности управление доступом осуществляется путем явной выдачи полномочий на проведение действий с каждым из объектов системы. Например, в модели Харрисона-Руззо-Ульмана [[4]] для этого служит матрица доступа, в которой определены права доступа

субъектов системы к объектам. Строки матрицы соответствуют субъектам, а столбцы – объектам. Каждая ячейка матрицы содержит набор прав, которые соответствующий субъект имеет по отношению к соответствующему объекту.

Как правило, создатель объекта обладает на него полными правами и может делегировать часть прав другим субъектам.

Дискреционный подход позволяет создать гораздо более гибкую схему безопасности, чем мандатный, но при этом он и гораздо более сложен в администрировании. С программной точки зрения его реализация очень проста, но при достаточно большом количестве объектов и субъектов система становится практически неуправляемой.

Частично решить эту проблему позволяет группировка пользователей и типизация объектов. Применение этих методов существенно уменьшает матрицу доступа и, соответственно, упрощает ее администрирование.

### **Ролевая модель**

В ролевой модели [[1]] операции, которые необходимо выполнять в рамках какой-либо служебной обязанности пользователя системы, группируются в набор, называемый «ролью». Например, операции по регистрации документов могут быть сгруппированы в роль «регистратор».

Каждый пользователь системы играет в ней одну или несколько ролей. Выполнение пользователем определенного действия разрешено, если в наборе его ролей есть нужная, и запрещено, если есть нежелательная.

В этой модели у объектов нет определенных хозяев. Вся информация расценивается как принадлежащая организации, владеющей системой. Соответственно, и роли пользователя внутри системы – это роли, которые он играет в данной организации. Как следствие, пользователю невозможно делегировать права на какой-то определенный объект. Либо у него есть доступ ко всем подобным объектам системы, либо нет.

Таким образом, преимуществом ролевой модели перед дискреционной является простота администрирования: назначение пользователей на роли и создание новых ролей не составляют никаких трудностей. Это позволяет рассматривать ролевую модель как наиболее подходящую для применения в прикладных программах. В то же время в ней есть ограничения,

которые в ряде случаев сильно затрудняют ее использование. Рассмотрим эти ограничения более подробно.

### **Ограничение: все роли глобальны**

Первое ограничение состоит в том, что пользователь принимает свои роли по отношению ко всей системе сразу. Соответственно, для системы нет разницы в правах между двумя пользователями, находящимися на одинаковой должности, даже если они занимают эти должности в разных отделах. Например, любой пользователь в роли «начальник отдела» имеет право управлять любым отделом своей организации, а это, конечно, неправильно.

Решением могло бы стать введение отдельных ролей «начальник отдела А», «начальник отдела Б» и т.п., что позволило бы нам решить проблему, не выходя за рамки ролевой модели. К сожалению, подобный вариант приносит гораздо больше проблем, чем решает.

Более правильным будет разбить все множество объектов системы на несколько подмножеств (доменов) и дать пользователям возможность играть разные роли в разных доменах системы. Такой подход применяется, например, в библиотеке Microsoft Authorization Manager [[6]].

### **Ограничение: отсутствие владельца объекта**

Второе препятствие перед использованием ролевой модели в ряде систем – это отсутствие в ней понятия владельца объекта. Другими словами, пользователь, создавший объект, не имеет на него никаких исключительных прав. Это вполне приемлемо для систем, поддерживающих, например, процесс купли-продажи, но перестает годиться, как только документы начинают содержать какие-то авторские материалы.

Зачастую для решения этой проблемы к объектам системы добавляют свойство «владелец», являющееся внешним по отношению к модели безопасности. Другой подход – ввести в ролевую модель элементы дискреционной и явным образом дать пользователю нужные права на созданный им объект.

Заметим, что оба эти варианта решают задачу, используя внешние по отношению к модели средства, и, соответственно, не снимают ограничений самой модели.

### **Ограничение: операции принадлежат ролям**

Казалось бы, группировка операций системы в роли, в рамках которых они выполняются, упрощает администрирование, но это снова верно не для всех типов систем. Предположим, что в нашей системе есть десять различных типов объектов, для каждого из которых определена операция «удалить». Тогда, если мы добавляем эту операцию в какую-либо из ролей, то любой пользователь, играющий эту роль, получает право удалять объекты любого типа. Очевидно, что это далеко не всегда является желательным поведением.

Можно попытаться решить эту проблему, введя десять различных операций, предназначенных для удаления объекта каждого из типов. Такое решение оказывается не очень удачным, если типов не десять, а, например, сто.

Проблема еще более усугубляется, если в системе возможны различные схемы доступа к разным объектам одного типа. Например, объекты неких типов могут быть или открытыми для публики, или совершенно секретными. Создавать действия «удалить публичный объект», «удалить секретный объект» и т.п. кажется совершенно неразумным.

На практике, при использовании ролевой модели в сложных системах, разработчики обычно не пытаются декларативно задавать схему доступа к объектам системы. Вместо этого процедура проверки встраивается в нужное место программы. При этом проверяются как сведения, предоставляемые модулем ролевой безопасности (т.е. роли, в которых выступает пользователь), так и любые другие сведения об объекте (владелец объекта, уровень секретности и т.п.).

Подобный подход затрудняет изменение схемы доступа, так как для этого нужно исправлять код процедуры проверки и перекомпилировать приложение.

### **Улучшенная ролевая модель (модель ForceField)**

ForceField – это разработанная нами модель управления доступом, которая позволяет создавать простые в администрировании политики безопасности. Она является существенно более мощной, чем ролевая модель, и при этом лишена ее основных недостатков.

### **Дерево объектов**

Мы обсуждали проблему диапазона действия ролей в базовой ролевой модели [[1]] и вариант ее решения с

использованием доменов. Недостаток этого решения в том, что домены нужно создавать вручную, и они явным образом не связаны ни с какими объектами системы. К тому же отсутствует иерархия доменов, а значит, наборы ролей пользователя в разных доменах приложения совершенно независимы. Это создает трудности, если пользователь должен играть определенную роль во всей системе сразу – его придется назначить на эту роль в каждом домене в отдельности.

В модели ForceField все объекты системы объединяются в единое дерево. У каждого объекта, кроме единственного корневого, есть один родительский объект, и любое количество дочерних. Роль может быть назначена пользователю в контексте любого объекта. При этом пользователь начинает играть назначенную роль во всей ветви дерева, которая образована этим объектом.

Таким образом, любой объект приложения образует домен, в который входит он сам и все его дочерние объекты. Его дочерние объекты образуют домены, являющийся подчиненными по отношению к домену родительского объекта. Список ролей, которые пользователь играет в определенном домене, состоит из ролей, назначенных ему в данном домене, плюс роли из доменов вверх по иерархии. Роли, назначенные пользователю в корневом домене, имеют глобальный характер, т.е. действительны в контексте каждого объекта приложения.

### **Роль «владелец»**

Вернемся еще раз к проблеме отсутствия владельцев у объектов в базовой ролевой модели. В качестве решения подойдет любой механизм, позволяющий выделить хозяина объекта и дать ему особые права на этот объект. В нашей модели для этого вводится роль «владелец», назначающаяся пользователям в контексте тех объектов, которыми они владеют.

Эта роль по своему поведению отличается от других ролей. Во-первых, только один пользователь может играть роль «владелец» в контексте какого-то определенного объекта. Во-вторых, объект не должен наследовать роль «владелец» от родителя, если в его собственном контексте такая роль кому-либо назначена.

Развивая эту идею, заметим, что роль может быть ограничена не только одним актером, но и большим их количеством. Таким образом, если роль ограничена п

пользователями, то в контексте любого объекта системы не больше  $n$  пользователей могут играть эту роль. При этом, очевидно, что в системе в целом у этой роли может быть больше  $n$  назначений.

В базовой модели подобные ограничения называются кардинальностью роли и определены как максимальное количество пользователей, которые могут играть эту роль в рамках всей системы.

### **Класс доступа**

Мы упоминали, что, хотя распределение операций по ролям кажется логичным, оно весьма затрудняет разработку схемы безопасности. Использование же сценариев для проверки прав доступа усложняет администрирование системы.

Для создания гибкой схемы безопасности без ручного программирования сценариев проверки, в ForceField введено понятие «класс доступа». Класс доступа содержит набор правил, задающих права выполнения определенных операций для определенных ролей. Порядок следования правил важен, так как при проверке поиск ведется сверху вниз до тех пор, пока не будет найдено правило, подходящее проверяемой ситуации. Правило из нашего примера подойдет, если будет запрошено разрешение на удаление объекта, а пользователь в контексте этого объекта будет администратором.

Каждому объекту системы ставится в соответствие ровно один класс доступа, а любой класс доступа может быть назначен произвольному количеству объектов. Это позволяет иметь в системе несколько разных схем доступа к объектам, не заставляя нас связывать эти схемы с типами или какими-то другими признаками объектов. Отметим также, что назначения классов никак не связаны с иерархией объектов: дочерний объект может иметь любой класс доступа, независимо от того, какой класс назначен родительскому объекту.

Класс может базироваться на другом классе, так что, если подходящего правила в классе нет, будет просмотрен базовый класс, потом его базовый класс и так далее. Если правило так и не будет найдено, то операция считается запрещенной. Этот механизм также служит для упрощения администрирования системы.

Перечислим некоторые возможные варианты распределения классов по объектам:



1. У всех объектов одного типа один и тот же класс доступа. Следует применять в системах, где различные типы отличаются друг от друга по правилам контроля доступа, но все объекты одного типа ведут себя одинаково.
2. Есть несколько классов доступа, которые могут быть назначены любому объекту, независимо от его типа. Например, классы, определяющие уровень секретности информации (публичная, для служебного пользования, совершенно секретная).
3. Есть всего один класс доступа, который назначается всем объектам.

### **Наследование резолюции сверху**

Существует также механизм наследования правил доступа от вышестоящих объектов. Подобный механизм часто встречается в файловых системах (дискреционная модель): файлы, лежащие в папке, могут не иметь своих собственных правил доступа, наследуя эти правила у папки.

В ForceField это реализуется следующим образом: в любом правиле класса кроме резолюций «разрешено» и «запрещено» можно использовать вариант «проверь родителя». В этом случае, для выдачи окончательного ответа будет проверено, можно ли данному пользователю выполнить запрашиваемое действие по отношению к родительскому объекту. Естественно, этот процесс может быть рекурсивным. Если по достижении корня иерархии объектов, явного разрешения или запрета так и не будет найдено, то действие считается запрещенным.

Чтобы продублировать описанное выше поведение файловой системы, достаточно создать класс, в котором объект наследует от родителя права на выполнение всех операций.

В то же время наша модель позволяет и такой вариант, когда права на часть операций наследуются, а для другой части задаются явным образом.

### **Гибридность модели ForceField**

Докажем, что предлагаемая нами модель позволяет реализовать в своих рамках функциональность базовых ролевой и дискреционной моделей.

## Реализация ролевой модели

Предположим, что в нашем приложении четыре объекта двух различных типов:  $A_1$ ,  $A_2$ ,  $B_1$ ,  $B_2$ . Существуют две операции, которые можно выполнить над объектом типа  $A$ , и одна для объектов типа  $B$ :  $op_{A1}$ ,  $op_{A2}$ ,  $op_{B1}$ . Также в системе зарегистрировано два пользователя:  $U_1$  и  $U_2$ .

Базовая ролевая модель предписывает нам ввести в систему роли и распределить операции между ролями. Введем следующие роли. Роль  $r_1$  включает в себя операции  $op_{A2}$  и  $op_{B1}$ , а роль  $r_2$  – операцию  $op_{A1}$ . Назначим на роли пользователей:  $U_1$  играет в системе роль  $r_2$ , а  $U_2$  – обе роли. Напомним, что пользователь, назначенный на определенную роль, имеет право выполнять операции, входящие в эту роль, по отношению к любому объекту системы.

Эта схема проиллюстрирована рисунком 1.

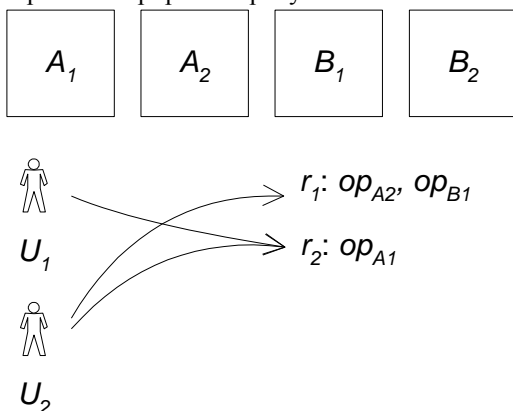


Рис. 1

Чтобы реализовать ее в нашей модели, необходимо сначала свести все объекты приложения в единую иерархию. Для этого достаточно добавить фиктивный корневой объект –  $root$  – и сделать объекты его прямыми потомками. Для того чтобы все роли были глобальными (как того требует базовая модель) мы должны будем назначать пользователей на них только в контексте корневого объекта, или, другими словами, в корневом домене  $d_{root}$ . Мы вводим роли  $r_1$  и  $r_2$  и назначаем на них пользователей.

После введения в систему операций остается только одна нерешенная проблема: необходимо, чтобы роль определялась

операциями, выполняемыми в ее рамках. Действительно, роль в модели ForceField, в общем случае, не соответствует этому требованию. Фактически она ничем не отличается от группы пользователей.

Решение заключается во вводе в систему единого для всех объектов класса доступа  $c_0$ , в котором операции, составляющие определенную роль, для этой роли явным образом разрешены. Очевидно, что это и есть нужный нам способ записи соответствия между ролями и операциями.

Принципиальная схема реализации приведена на рисунке 2.

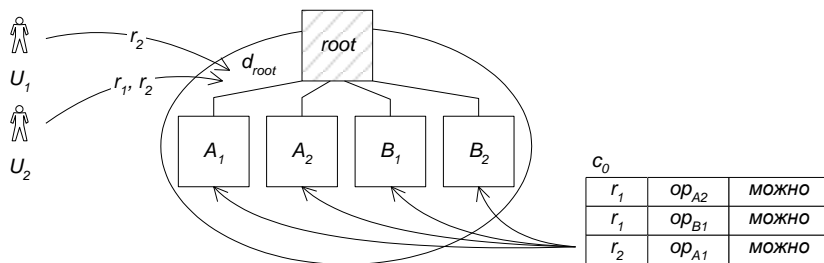


Рис. 2

### Реализация дискреционной модели

Для приложения, описанного в предыдущей задаче, система контроля доступа по модели Харрисона-Руззо-Ульмана будет подобна схеме на рисунке 3. В ней столбцы соответствуют объектам, строки – пользователям. В ячейках прописаны индивидуальные права пользователя на соответствующий объект. Отметим, что, хотя в этом и нет большой необходимости, в таблице явным образом запрещены операции, не имеющие смысла для соответствующих объектов.



	$A_1$	$A_2$	$B_1$	$B_2$
	$op_{A1}$ - да	$op_{A1}$ - да	$op_{A1}$ - нет	$op_{A1}$ - нет
	$op_{A2}$ - нет	$op_{A2}$ - нет	$op_{A2}$ - нет	$op_{A2}$ - нет
$U_1$	$op_{B1}$ - нет	$op_{B1}$ - нет	$op_{B1}$ - нет	$op_{B1}$ - нет
	$op_{A1}$ - да	$op_{A1}$ - да	$op_{A1}$ - нет	$op_{A1}$ - нет
	$op_{A2}$ - да	$op_{A2}$ - да	$op_{A2}$ - нет	$op_{A2}$ - нет
$U_2$	$op_{B1}$ - нет	$op_{B1}$ - нет	$op_{B1}$ - да	$op_{B1}$ - да

Рис. 3

Для реализации этой модели в ForceField нужно создать четыре отдельных класса доступа ( $c_{A1}$ ,  $c_{A2}$ ,  $c_{B1}$  и  $c_{B2}$ ) и назначить их соответствующим их объектам. В составляющих классы правил разрешения будут даваться не ролям, а непосредственно пользователям. Необходимо подчеркнуть, что возможность указывать в правиле не роль, а пользователя, следует использовать только в крайних случаях, так как это может привести к созданию плохо управляемой политики безопасности.

В классы не включены правила, запрещающие не имеющие смысла операции, т.к. все явным образом не разрешенное автоматически считается запрещенным.

Результирующая схема приведена на рисунке 4.

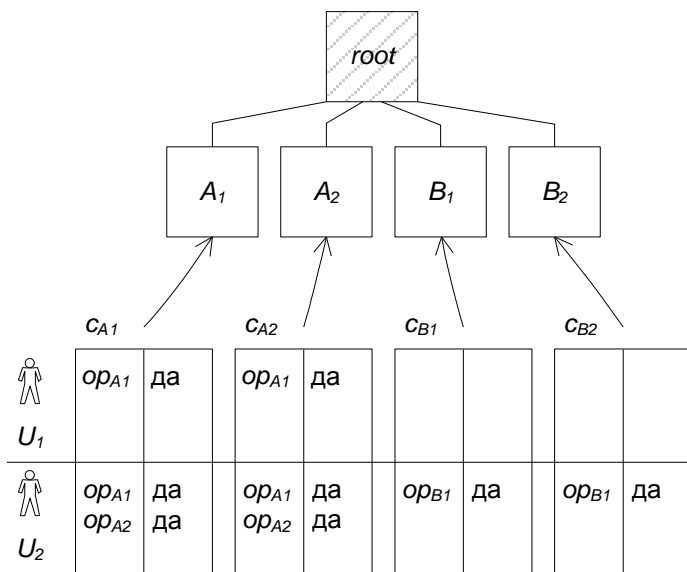


Рис. 4

На схеме не показан корневой домен и назначения пользователей на роли. Это связано с тем, что рассматриваемая модель является слишком простой, и роли в ней не используются. В таком виде модель не годится для большинства реальных применений, поэтому расширим ее, добавив типы объектов и группы пользователей.

В нашем приложении изначально есть два типа объектов, поэтому введем их в систему, связав с классами доступа. Таким образом, количество классов безопасности уменьшается до двух: класс безопасности для объектов типа А ( $C_A$ ) и класс для объектов типа В ( $C_B$ ).

Для группировки пользователей мы можем использовать роли, назначаемые пользователям в корневом домене. Поэтому введем две роли:  $g_1$  и  $g_2$ . Изменим правила в классах доступа, с тем чтобы они выдавали разрешения группам пользователей (т.е. ролям), а не каждому пользователю в отдельности.

На рисунке 5 приведена схема реализации дискреционной модели, оптимизированной за счет типизации объектов и введения групп пользователей. В нашей модели можно реализовать и другие способы оптимизации. Например, наследование прав по аналогии с файловой системой легко реализуется за счет специального класса доступа на дочернем объекте. Он должен определять собственную схему доступа объекта и дополнять ее правилом «любая роль – любая операция – как у родителя».

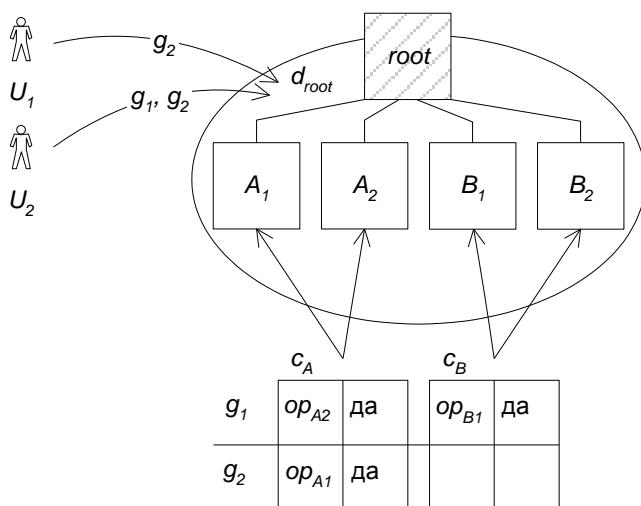


Рис. 5

### Введение элементов мандатной модели

Самой первой из рассмотренных нами классических моделей была мандатная модель. В дальнейшем мы не уделяли ей достаточного внимания, так как она является весьма экзотичной, и крайне редко применяется в реальных приложениях. В связи с этим мы не ставили перед собой задачу охвата и этой модели, но наметили пути, по которым это может быть сделано.

Во-первых, классам доступа нужно назначить уровни секретности. Так как каждый объект приложения проассоциирован с определенным классом доступа, это автоматически назначит уровни секретности и всем объектам.

Во-вторых, мы должны назначить уровни допуска пользователям системы. В соответствии с моделью, пользователь будет иметь право читать документы с уровнем секретности не выше его собственного, и изменять документы с уровнем не ниже.

В-третьих, необходимо разделить все операции системы на две группы: группу чтения и группу записи. Тогда, при проверке на допустимость операции, мы будем точно знать, какое правило применить.

Наконец, модифицируется процедура проверки прав. Если у объекта и пользователя разные уровни, то мы проводим проверки по стандартным принципам мандатной модели. Если уровень одинаковый, то применяем обычные правила нашей модели.

Заметим, что нужно будет еще тщательно обдумать желаемое поведение системы в ситуации, когда по мандатной модели операция разрешена, а по модели ForceField — запрещена. Ответ на этот вопрос определит направленность системы безопасности. Нужно решить, что приоритетней: полная свобода пользователям с высоким уровнем или ограничение пользователей с низким.

## **Заключение**

Мы показали, что предлагаемая модель контроля доступа объединяет в себе базовые модели. Что более важно, она расширяет их возможностью назначения пользователей на роли в контексте любого объекта системы. Поэтому множество ролей, которые пользователь играет в некий момент времени, не является одним и тем же для всех объектов приложения, а пополняется новыми ролями по мере спуска вглубь по объектной иерархии.

За счет этого появляется возможность максимально естественным образом ограничить область действия выданных

пользователю полномочий определенной частью приложения. Это позволяет уменьшить необходимое количество ролей и упростить схемы доступа к объектам.

Таким образом, наша модель позволяет создавать легкие в администрировании политики безопасности, обладая, в то же время, необходимыми возможностями по ограничению несанкционированного доступа к объектам приложения.

## **Литература**

- [1] Leonard J. LaPadula and D. Elliott Bell. Secure Computer Systems: A Mathematical Model // MITRE Corporation Technical Report 2547. Volume II. 31 May 1973.
- [2] Зегжда Д.П. Общая схема мандатных моделей безопасности и ее применение для доказательства безопасности систем обработки информации // Проблемы информационной безопасности. Компьютерные системы. СПбГТУ. 2000. 2.
- [3] Степанов П.Г. Принципы управления доступом к ресурсам в защищенной ОС «Феникс» // Проблемы информационной безопасности. Компьютерные системы. СПбГТУ. 1999. 4.
- [4] M. Harrison, W. Ruzzo, J. Uhlman Protection in operating systems // Communications of the ACM. 1976.
- [5] Баранов А.П. Зегжда Д.П., Зегжда П.Д., Ивашко А.М., Корт С.С. Теоретические основы информационной безопасности (Дополнительные главы). СПб.: СПбГТУ. 1998.
- [6] Mohan Rao Cavale. Role-Based Access Control Using Windows Server 2003 Authorization Manager. <http://msdn.microsoft.com/library/en-us/dnnetserver/html/AzManRoles.asp>



# **The process of developing hardware and software systems based on visual modeling using SysML/UML**

**Dmitry Ryzhov**  
**SWD Software Ltd.**  
**email: d.ryzhov@swd.ru**

**Denis Ivanov**  
**Ай Ти Консалтинг.**  
**email: denis.ivanov@it-**  
**konsulting.spb.ru**

## **Abstract**

More and more, systems engineers are turning to the System Modeling Language (SysML) to specify and design their embedded systems. This has many advantages, including verifiability and ease of passing off information to other engineering disciplines, particularly software and hardware. This paper describes a SysML-based process that systems engineers can use to capture requirements and specify architecture. The process is function-driven and is based heavily on the identification and elaboration of operational contracts, a message-based interface communication concept. The process described is developed by Telelogic and actively being used in real projects for embedded systems development based on Model-Driven Development (MDD) approach.

**Keywords:** System Modeling Language (SysML), embedded systems, system&software integrated development process, functional decomposition, model verification and validation

# **Процесс Разработки Программно-Аппаратных Систем на Основе Визуального Моделирования с Использованием SysML/UML**

Дмитрий Рыжов  
SWD Software Ltd.  
email: d.ryzhov@swd.ru

Денис Иванов  
Ай Ти Консалтинг.  
email: denis.ivanov@it-konsulting.spb.ru

## **Тезисы**

Все большее число системных инженеров начинают использовать язык моделирования SysML для специфицирования и проектирования программно-аппаратных систем. Это дает им много преимуществ, в том числе возможности верификации и валидации моделей систем и облегчение передачи информации представителям других инженерных дисциплин, в частности разработчикам программного и аппаратного обеспечения. Данная статья содержит описание процесса разработки систем, основанного на SysML, который системные инженеры могут использовать для спецификации требований и определения архитектуры систем. В основе процесса лежит функциональная декомпозиция, основанная на определении и дальнейшем уточнении операционных контрактов, а также концепция взаимодействия на основе обмена сообщениями. Описываемый процесс разработан компанией Telelogic и активно используется во множестве реальных проектов по разработке программно-аппаратных систем на основе визуального моделирования (MDD).

**Keywords:** язык моделирования SysML, программно-аппаратные системы, интегрированный процесс разработки систем и ПО, функциональная декомпозиция, верификация и валидация модели

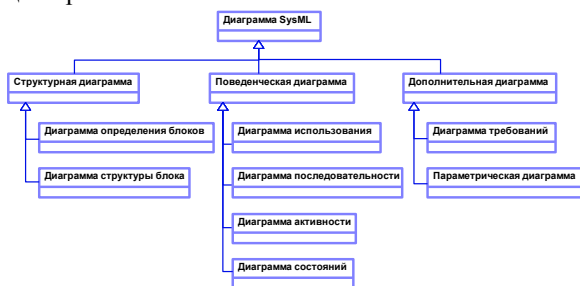
## Введение

Многие годы разработчики программного обеспечения с успехом применяли язык UML для разработки программного обеспечения на основе визуального моделирования. Было предпринято несколько попыток применить UML и лежащий в его основе объектно-ориентированный подход к разработке систем в целом с целью унификации всего процесса разработки. Однако многие системные инженеры продолжают использовать классические методы структурного анализа и соответствующие артефакты для определения требований к системе и дальнейшего ее проектирования. Одна из возможных причин такого положения дел состоит в том, что разработка систем определяется в большей степени функциональными требованиями. Формулировки в терминах функциональности системы при ее проектировании все еще рассматриваются как наиболее естественными большинством специалистов инженерных дисциплин, вовлекаемых на ранних этапах разработки систем (схемотехников, конструкторов, маркетологов и, конечно, заказчиков). Учитывая вышесказанное, единственный путь унификации всего процесса разработки заключается в расширении UML для получения возможности разработки систем на основе функциональной декомпозиции и определении процесса, позволяющего осуществлять плавную передачу результирующих артефактов разработчикам программного обеспечения, использующим UML. Для достижения этой цели OMG (Object Management Group) сформировала консорциум для разработки языка SysML (Systems Modeling Language).

На Рис.3 изображены диаграммы языка SysML. Язык SysML является диалектом языка UML. Часть диаграмм языка UML не вошли в язык SysML, но появились две новых. Другие диаграммы UML были дополнены и частично изменены. Базовым элементом модели стал блок, а не класс. Диаграмма классов и диаграмма структуры UML получили новые названия.

В статье описывается процесс, основанный на языке моделирования SysML, который системные инженеры могут использовать для спецификации требований и определения архитектуры систем. Рассматриваемый процесс основан на возможности среды разработки Telelogic Rhapsody по

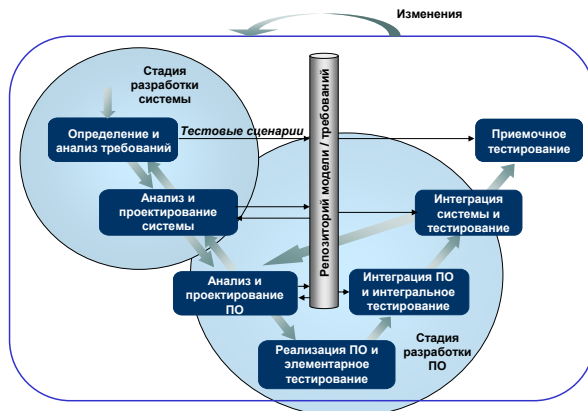
исполнению моделей, как средство для верификации и валидации требований.



**Рис.3** Диаграммы языка SysML

### Обзор процесса

На Рис.4 с использованием классической “V” диаграммы показан интегрированный процесс, совмещающий в себе разработку систем и программного обеспечения. Левая наклонная часть буквы “V” отображает нисходящий процесс проектирования, в то время как правая наклонная часть отображает восходящие этапы интеграции, начиная с элементарных тестов и заканчивая приемкой конечной системы. Используя нотацию состояний, итеративная природа процесса отображается в виде “высокоуровневого перехода”, возникающего при появлении изменений. Стадия разработки систем характеризуется последовательным нисходящим потоком работ, включающим анализ требований, анализ системы и проектирование ее архитектуры. Поток работ для стадии разработки программного обеспечения характеризуется итеративными инкрементными циклами, содержащими этапы анализа, проектирования, реализации, а также этапы интеграции и тестирования различного уровня.



**Рис.4 Интегрированный процесс разработки систем и программного обеспечения**



**Рис. 5 Процесс разработки систем, основанный на SysML**

На Рис.4 также отображен факт создания и повторного использования на всех этапах проектирования тестовых сценариев, определяемых на основе исходных требований. Эти сценарии используются на этапах интеграции и тестирования, а также для регрессионного тестирования при внесении изменений в систему.

Перечислим ключевые цели процесса разработки систем:

- Определение требуемой функциональности системы и источников происхождения требований
- Определение состояний и режимов работы системы
- Определение подсистем и их компонентов и привязка к ним требований и реализуемой функциональности

Для процесса моделирования эти цели подразумевают движение сверху вниз, начиная с абстракций самого высокого уровня. На Рис. 5 приведена схема процесса разработки систем, основанного на SysML. Для каждой фазы процесса показаны основные этапы, а также входные данные и создаваемые артефакты. В следующих разделах приводится их описание, а также определение последовательности шагов для каждого из этапов.

## **Фаза анализа требований**

### **3.1. Этап определения требований**

Фаза анализа требований начинается с анализа имеющейся информации. На основе требований заказчика создаются системные требования, которые определяют, что система должна делать (функциональные требования) и как хорошо она должна это делать (требования к качеству).

### **3.2. Этап определения вариантов использования**

После того как требования к системе становятся понятны они группируются по вариантам использования. Один вариант использования описывает конкретный операционный аспект системы (поток операций). Он специфицирует поведение, которое видно пользователю, и определяет поток сообщений между пользователем и системой. При этом внутренняя структура системы не специфицируется никоим образом (представление "черного ящика"). Варианты использования могут быть структурированы иерархически. Для отображения

множества вариантов использования применяется диаграмма использования.

Определенные варианты использования связываются с требованиями к системе, после чего производится проверка полноты покрытия требований элементами модели.

### **Фаза функционального анализа**

На фазе функционального анализа для каждого определенного варианта использования создается отдельная модель системы “черного ящика” и относящиеся к ним требования верифицируются и валидируются путем исполнения модели. При этом используется подход, основанный на обмене сообщениями:

- Структура системы описывается с помощью диаграмм структуры SysML, в качестве базовых элементов используются блоки, для которых определяются порты и предоставляемые / требуемые интерфейсы.
- Взаимодействие между блоками осуществляется путем обмена сообщениями (запроса сервисов).
- Функциональность блоков определяется с использованием операционных контрактов (сервисов), например, операция1(),..., операция4()
- Функциональная декомпозиция осуществляется через декомпозицию операционных контрактов.

На Рис.6 показано как связаны между собой артефакты SysML, создаваемые для каждого варианта использования в процессе функционального анализа.

Анализ варианта использования “черного ящика” начинается с определения окружения для варианта использования. Для этого используется такой артефакт SysML как структурная диаграмма. Элементами этой диаграммы являются блоки, моделирующие действующих лиц и саму систему.

На следующем шаге анализа варианта использования определяются сценарии “черного ящика”. Один сценарий описывает определенную последовательность взаимодействия в рамках варианта использования. Он детализирует поток сообщений между действующими лицами и системой в рассматриваемом варианте использования, а также результирующее поведение (операционные контракты) на сторонах получателей сообщений. С использованием SysML сценарии определяются графически на диаграммах последовательности. Линии жизни на диаграмме

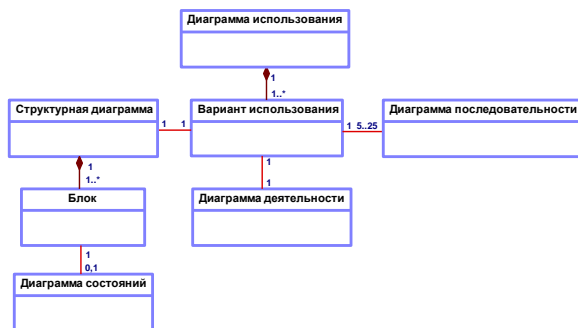
последовательности "черного ящика" относятся к действующим лицам и системе.

Как только набор существенных сценариев определен, полученные сценарии объединяются в единое описание последовательности действий для варианта использования. Для этой цели используется такой артефакт SysML как диаграмма деятельности. Каждое действие на диаграмме деятельности соответствует операционному контракту на диаграмме последовательности. Диаграмма деятельности "черного ящика" играет важную роль на этапе проектирования архитектуры системы.

Основываясь на информации, определенной на диаграммах последовательности и диаграмме деятельности "черного ящика", для блоков структурной диаграммы определяются порты и связанные с ними интерфейсы.

Следующим шагом в анализе варианта использования "черного ящика" является описание поведения системы на основе состояний. Для этого используется такой артефакт SysML как диаграмма состояний. Диаграмма состояний отображает состояния и режимы работы системы, а также логику их изменения при возникновении внешних воздействий. Создание диаграммы состояний для варианта использования производится на основе определенных на предыдущих шагах сценариев.

На следующем шаге производится верификация и валидация модели варианта использования и связанных с ним требований. Верификация и валидация осуществляются посредством исполнения модели с использованием определенных сценариев "черного ящика" как основы для генерации



**Рис.6 Артефакты SysML, создаваемые в процессе функционального анализа**



входных воздействий для исполняемой модели. Следуя определенным выше ключевым целям процесса разработки систем, необходимо отметить, что основной фокус при этом направлен больше на анализ получаемых последовательностей вызовов, чем на реализуемую на их основе функциональность. После того как созданы модели для всех вариантов использования определенные в них операционные контракты для системы объединяются в одном системном блоке. В конце фазы функционального анализа, системный блок содержит верифицированные и валидированные операционные контракты, соответствующие функциональным требованиям к системе.

## **Фаза проектирования архитектуры**

### **5.1. Этап проектирования архитектуры системы**

Целью этапа проектирования архитектуры системы является привязка верифицированных и валидированных операционных контрактов к элементам физической архитектуры системы. Привязка является итеративным процессом к которому привлекаются эксперты из различных инженерных дисциплин. Они могут проанализировать различные архитектурные концепции, с учетом требований к производительности, безопасности и стоимости, определенные на этапе анализа требований.

Проектирование архитектуры системы начинается с определения физических подсистем. Для этого используется такой артефакт SysML как структурная диаграмма. Элементами модели являются блоки действующих лиц и системный блок. Частями системного блока являются физические подсистемы, определенные на основе выбранной архитектуры.

Следующие несколько шагов выполняются итеративно для каждого варианта использования.

Определенные ранее операционные контракты для системы в конкретном варианте использования привязываются к физическим подсистемам с использованием диаграммы деятельности "белого ящика". Эта диаграмма является практически копией диаграммы деятельности "черного ящика". Единственное различие состоит в том, что данная диаграмма разбита на разделы, каждый из которых соответствует физической подсистеме. На основе определенной архитектуры, операционные контракты системы размещаются на диаграмме в

разделах соответствующих подсистем. Важным постулатом для данной привязки является то, что изначальные связи (последовательность действий) между операционными контрактами сохраняются.

Помимо диаграммы деятельности "белого ящика" для рассматриваемого варианта использования создаются диаграммы последовательности "белого ящика". Они представляют собой декомпозицию ранее определенных диаграмм последовательности "черного ящика" и используются для определения интерфейсов физических подсистем. На диаграммах последовательности "белого ящика" линия жизни системы расщепляется на множество линий жизни соответствующих подсистем. В соответствии с привязкой сделанной на диаграмме деятельности "белого ящика", операционные контракты подсистем перемещаются на соответствующие линии жизни. Для сохранения изначальной определенной последовательности действий может возникнуть необходимость определить дополнительные запросы сервисов между физическими подсистемами. Они дополняют интерфейсы между подсистемами.

На основе полученных сценариев "белого ящика" между физическими подсистемами определяются связи и соответствующие порты и интерфейсы.

Шаги определенные выше повторяются для всех вариантов использования.

После обработки всех вариантов использования для каждой физической подсистемы определяется поведение с использованием диаграммы состояний. Диаграммы состояний создаются с использованием полученных сценариев "белого ящика"

Корректность и полнота модели архитектуры системы проверяется путем ее исполнения. После того, как функциональность модели верифицирована, может быть проведен анализ архитектуры на соответствие требованиям к производительности и безопасности с привлечением соответствующих инженерных методов.

## **5.2. Этап проектирования архитектуры подсистем**

На данном этапе основной фокус делается на определении способа реализации привязанных к подсистемам операционных контрактов.

Последовательность шагов, которые требуется для этого сделать, аналогична этапу проектирования архитектуры системы. При этом проектирование архитектуры подсистем производится отдельно для каждой подсистемы. Ниже определена последовательность шагов, которая выполняется итеративно для каждого варианта использования подсистемы.

На первом шаге принимается решение о том, какие операционные контракты физической подсистемы следует реализовать в аппаратуре (механически или с использованием электроники), а какие с помощью программного обеспечения. Для этого используются расширенные диаграммы активности "белого ящика". Для операционных контрактов, которые затрагивают более чем одну инженерную дисциплину, потребуется дополнительный анализ. В данном анализе могут участвовать эксперты из различных инженерных дисциплин, работающие над подсистемой.

Для каждого сценария "белого ящика" рассматриваемого варианта использования, линии жизни физической подсистемы расщепляются на линии жизни аппаратных и/или программных компонентов. В соответствии с выбранной архитектурой, операционные контракты помещаются на соответствующие линии жизни компонентов, а определенная ранее последовательность взаимодействия между подсистемами устанавливается через соответствующие запросы сервисов между частями этих подсистем - компонентами.

На основе полученных расширенных сценариев "белого ящика" определяются порты и интерфейсы компонентов подсистем.

Шаги определенные выше повторяются для всех вариантов использования физической подсистемы.

После этого для каждого компонента физической подсистемы определяется поведение с использованием диаграммы состояний.

Полученная модель размещения проверяется путем регрессионного тестирования.

По окончании этапа проектирования архитектуры подсистем модель размещения содержит операционные контракты, привязанные к аппаратным и программным компонентам, и связанные с изначальными требованиями.

Так как диаграммы SysML являются подмножеством диаграмм UML, это дает возможность плавного перехода к разработке программного обеспечения с использованием UML. Для перехода на последующую стадию разработки программного и аппаратного обеспечения, для каждой физической подсистемы

на основе модели размещения генерируются следующие документы:

- Спецификация требований для аппаратного и программного обеспечения
- Описание логических интерфейсов между компонентами
- Тестовые сценарии для подсистем и их компонентов полученные на основе сценариев вариантов использования уровня системы.

## **Заключение**

В этой статье продемонстрировано, что использование языка SysML позволяет унифицировать процесс разработки систем на основе визуального моделирования с применением функциональной декомпозиции. SysML может рассматриваться в качестве “диалекта” языка UML, понятного как для системных инженеров, так и для разработчиков ПО. Основываясь на текущей спецификации SysML (версия 1.0), может быть определен интегрированный процесс разработки систем и программного обеспечения, позволяющий осуществлять плавный переход между ними.

## **Литература**

[1] Hans-Peter Hoffman, SysML-Based Systems Engineering Using a Model-Driven Development Approach, Telelogic whitepaper, 1 July 2008, (<http://modeling.swd.ru>)

[2] B.P. Douglass, Real Time UML Workshop for Embedded Systems, Book, Elsevier, 2007

[3] Tim Weilkiens, Systems Engineering with SysML/UML: Modeling, Analysis, Design, Book, Elsevier, 2006

[4] B.P. Douglass, Real-Time UML Advances in the UML for Real-Time Systems, Book, Addison-Wesley, 2004

[5] B.P. Douglas, Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns, Book,

## **Заключение**

В этой статье продемонстрировано, что использование языка SysML позволяет унифицировать процесс разработки систем на основе визуального моделирования с применением функциональной декомпозиции. SysML может рассматриваться в качестве “диалекта” языка UML, понятного как для системных инженеров, так и для разработчиков ПО. Основываясь на текущей спецификации SysML (версия 1.0), может быть определен интегрированный процесс разработки систем и программного обеспечения, позволяющий осуществлять плавный переход между ними.

## **Литература**

[1] Hans-Peter Hoffman, SysML-Based Systems Engineering Using a Model-Driven Development Approach, Telelogic whitepaper, 1 July 2008, (<http://modeling.swd.ru>)

[2] B.P. Douglass, Real Time UML Workshop for Embedded Systems, Book, Elsevier, 2007

[3] Tim Weilkiens, Systems Engineering with SysML/UML: Modeling, Analysis, Design, Book, Elsevier, 2006

[4] B.P. Douglass, Real-Time UML Advances in the UML for Real-Time Systems, Book, Addison-Wesley, 2004

[5] B.P. Douglas, Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns, Book, Addison-Wesley, 1999

# **Experience in creation and implementation of integrated software development process automation system**

**Grachev Anton**  
**Luxoft Professional**  
**Moscow, Russia**

**email: [Agrachev@luxoft.com](mailto:Agrachev@luxoft.com)**

**Gavrilov Evgeny**  
**Luxoft Professional**  
**Moscow, Russia**

**email: [Egavrilov@luxoft.com](mailto:Egavrilov@luxoft.com)**

## **Abstract**

The goal of the article is a LUXOFT experience presentation in development and implementation of LUXproject integrated development environment for software engineering process automation. Major issues of process automation tools are lack of integration, geographic team distribution, using different types of software engineering processes (therefore different sets of tools).

LUXproject is a “process environment” based on Atlassian products: JIRA and Confluence. Other tools required for process support could be integrated (or are already integrated) as well. Currently the solution implements RUP model (with CMMI support) and Agile model .

From the user’s perspective LUXproject incorporates integrated products in a single Web interface and allows using the functionality of each integrated component in any projects. UI integration leads to a number of issues in keeping the uniform UI style as well as usability issues. Known solutions are described in the article.

Functional integration allows administering all components from a single place, activities monitoring, discovering artifacts links for future traceability and managing activities. System architecture based on “message bus” pattern and unified domain language. This will allow developing new functionality in future releases effectively.

Major implementation scenarios are presented: implementation of the process that is already used within an existing project as well as implementation of the predefined process templates to the new projects

**Keywords:** Automatization of software development process

# **Опыт создания и внедрения интегрированной системы автоматизации процессов разработки программного обеспечения.**

**Грачев Антон**  
**Люксофт Профешнл**  
**Москва, Россия**  
**email: Agrachev@luxoft.com**

**Гаврилов Евгений**  
**Люксофт Профешнл**  
**Москва, Россия**  
**email: Egavrilov@luxoft.com**

## **Тезисы**

Целью статьи является представление опыта компании LUXOFT в создании и внедрении интегрированной системы для автоматизации процессов разработки программного обеспечения LUXproject. Описываются проблемы автоматизации процессов разработки: отсутствие интеграции, географическая распределенность команды и потребность использовать разные процессы разработки (следовательно, разные инструменты автоматизации).

LUXproject является «процессной оболочкой», которая базируется на продуктах Atlassian JIRA и Atlassian Confluence, интегрируя также и другие приложения, необходимые для поддержки процесса. В настоящий момент, LUXproject реализует модели процессов RUP (с поддержкой CMMI) и Agile.

С точки зрения пользователя, LUXproject объединяет интегрированные продукты в едином web-интерфейсе, предоставляя возможность пользоваться функциональностью каждого продукта во всех проектах. Это порождает проблемы поддержки единого стиля приложения и проблемы usability, решения для которых предложены в докладе.

Функциональная интеграция позволяет администрировать все компоненты из одной точки а также производить мониторинг активностей, отслеживать связи артефактов процесса и управлять активностями. Архитектура системы построена на основе шины сообщений и единого языка доменной модели, что позволяет быстро расширять функциональность продукта в дальнейшем.

Рассматриваются сценарии внедрения системы, как с учетом существующих работающих процессов в компании (отделе), так и для новых проектов с готовыми предустановленными в системе шаблонами процессов.

**Keywords:** автоматизация процессов разработки.

## 1. Введение

Целью данной статьи является представление опыта компании LUXOFT в создании и внедрении интегрированной системы для автоматизации процессов разработки программного обеспечения.

Не секрет, что для многих компаний, занимающихся разработкой и внедрением программного обеспечения, стала стандартной практикой автоматизация процессов разработки. Для большинства компаний такие процессные области как, «управление проектами», «управление требованиями» «тестирование», «конфигурационное управление», уже немислимы без автоматизации.

Однако, несмотря на довольно серьезную автоматизацию, существуют ряд проблем, которые не позволяют говорить о создании единой автоматизированной среды разработки и внедрения.

Для примера рассмотрим несколько ситуаций, иллюстрирующих острые проблемы автоматизации.

Первая из рассмотренных нами проблем, заключается в сложности быстрого получения информации о различных данных в проекте и ее сопоставления в едином визуальном «интерфейсе». Проиллюстрируем эту проблему на примере - в некоторой компании применяется программа для тестирования. Разработчик, получив данные о дефектах в коде разрабатываемого продукта, сталкивается с проблемой отождествления дефекта с описанием соответствующего требования, так как в системах тестирования в лучшем случае заносятся заголовки требований, а за подробным описанием приходится «залезать» в дебри уже программы по сбору требований.

Вторая проблема заключается в автоматизации распределенной разработки. Команды проекта, находящиеся в различных географических точках, могут испытывать проблемы, как с доступом к единой проектной базе, так и сложности с использованием систем у заказчика по политикам безопасности.

Третья проблема может заключаться в наличии различных подходов к разработке. В одном проекте используют Agile практики, в другом процессы RUP, соответственно множится и программное обеспечение для автоматизации процессов разработки, что влечет увеличение затрат Компании на автоматизацию.



Решением вышеописанных проблем, может стать единая интегрированная система разработки и управления проектом основанная на WEB решении для распределенного доступа.

## **2. Опыт компании LUXOFT в создании и внедрении интегрированной системы**

Компания LUXOFT, в том числе и для преодоления вышеописанных проблем, создала свою интегрированную систему автоматизации процесса разработки - LUXproject ®.

При разработке данной системы были поставлены следующие цели:

- Комплексная поддержка жизненного цикла разработки программного обеспечения (от заключения контракта до поддержки);
- Поддержка распределенной разработки
- Поддержка различных систем разработки

По сути LUXproject является «процессной оболочкой», которая базируется на Atlassian JIRA и Atlassian Confluence. Благодаря данной «процессной оболочке» существует возможность интеграции с различными программными продуктами, как линейки продуктов компании Atlassian, так и продуктами других компаний.

В настоящий момент LUXproject поддерживает две модели процессов разработки: одна основана на процессах RUP (и поддерживает модель CMMI), вторая - на Agile практиках.

Для модели, основанной на процессах RUP, в настоящий момент реализован следующий процессный функционал:

- Управление задачами;
- Управление рисками;
- Формирование отчетности;
- Управление требованиями;
- Управление изменениями;
- Управление сборкой;
- Управление тестированием (включает управление test cases и дефектами);
- Управление качеством (процессные аудиты и анализ статистических данных);
- Управление конфигурацией;

- Управление коммуникациями (возможность вести базу знаний);

Для модели, основанной на практиках Agile реализован следующий функционал:

- Управление задачами (в части - ведение бэклога продукта, управление релизами, ведение бэклога итерации, персональный план работ);
- Управление рисками;
- Управление дефектами;
- Управление коммуникациями (Scrum and retrospective meetings);
- Управление конфигурацией;

Основными пользователями системы являются, как исполнители проекта, так и заказчик.

Система поддерживает ролевой доступ, то есть каждый участник проектной команды в зависимости от проектной роли имеет тот или иной доступ к функционалу системы.

### **3. Интеграция интерфейсов пользователя**

С точки зрения пользователя, LUXproject объединяет интегрированные продукты в едином web-интерфейсе, предоставляя возможность пользоваться функциональностью каждого продукта во всех проектах.

При интеграции нескольких web-приложений в единую систему встает вопрос о том, как организовать объединение пользовательских интерфейсов, чтобы, с одной стороны, получить всю функциональность приложений, с другой – создалось ощущение единства стиля и целостности всей системы.

Первая задача интеграции заключается в введении общих навигационных элементов (header, footer) и единого стилового оформления на всех страницах системы.

Для решения поставленной задачи исследовались следующие подходы:

1. Использование существующих порталных решений позволяет относительно быстро интегрировать интерфейсы приложений. Но исследования показали, что они достаточно

тяжелы и требуют адаптации приложений к использованию в виде портлетов.

2. Использование комбинации паттернов “Proxy” и “Decorator” позволяет перехватить контент приложения, обработать его (добавив и изменив необходимые элементы) и выдать конечному пользователю. Минус подхода – перехват и фильтрация контента достаточно дорога по ресурсам и существенно влияет на быстродействие системы; наложение аспекта-декоратора на все страницы может вызвать большое количество разнообразных ошибок, т.е. требует большого объема тестирования. Плюсы – есть широкие возможности по контролю контента интегрируемого приложения, которые можно использовать для функциональной интеграции. Изменение оформления приложений достигается добавлением css-стилей в шапку в декоратор. В качестве реализации Proxy используется ProxyHttpServletRequest, декорирование обеспечивает фреймворк Sitemesh (<http://www.opensymphony.com/sitemesh>). Данный подход используется в LUXproject.

3. Поставленную задачу можно решить, внедрив в каждое приложение общие элементы (header, footer). Причем конечные пользователи будут ходить на те же приложения, что и раньше, но ссылки в header-е будут вести на разные приложения в интегрируемой системе. Этот подход имеет существенное преимущество в скорости доступа (фактически, интеграция почти не влияет на время отклика). Основной недостаток – необходимость изменять исходный код интегрируемого приложения, что не всегда представляется возможным. По сравнению с вариантом 2, теряется возможность контроля над интегрируемыми приложениями из одного (центрального) приложения. Этот подход был использован в продукте Atlassian JIRA Studio (<http://jira.com>).

После успешной реализации описанных задач, начинают возникать проблемы другого уровня сложности. Каждое приложение имеет собственную модель работы пользовательского интерфейса: названия и типовое расположение кнопок и других элементов управления, горячие клавиши, расположение и состав меню. Помимо этого, возможны опасные терминологические конфликты, когда под

одними и теми же названиями в интегрируемых системах скрываются совершенно разные понятия. Причем все подходы абсолютно оправданы в конкретных приложениях (документная модель Confluence, ориентированная на контент, и табличная модель JIRA), но сбивают с толку пользователей, работающих в них всех, практически одновременно. Данная проблема встает все острее при развитии функциональной интеграции, когда все больше сценариев использования выполняется в нескольких компонентах, заставляя пользователя переключаться с одной модели интерфейса на другую.

Предлагается два пути решения проблемы. Первый путь – дать пользователю возможность четко осознавать, когда он находится в одном приложении, а когда уже переключился в другое, как это сделано в Atlassian JIRA Studio при помощи закладок. Это не решает проблемы, но меньше запутывает пользователя и позволяет быстрее понять модель взаимодействия приложений. Второй путь – выносить большую часть функционала в одно приложение, используя другие в качестве движка. Например, если требования ведутся как wiki-страницы + JIRA items, возможно реализовать все операции с требованиями не выходя из wiki, используя JIRA только как движок Workflow и для формирования отчетов. В последнем случае, аналитику не придется переключаться между приложениями в большинстве случаев. Этот подход постепенно реализуется в LUXproject, где Wiki-движок Confluence используется как основа для интерфейса интеграции всех компонент (см. рис 1).

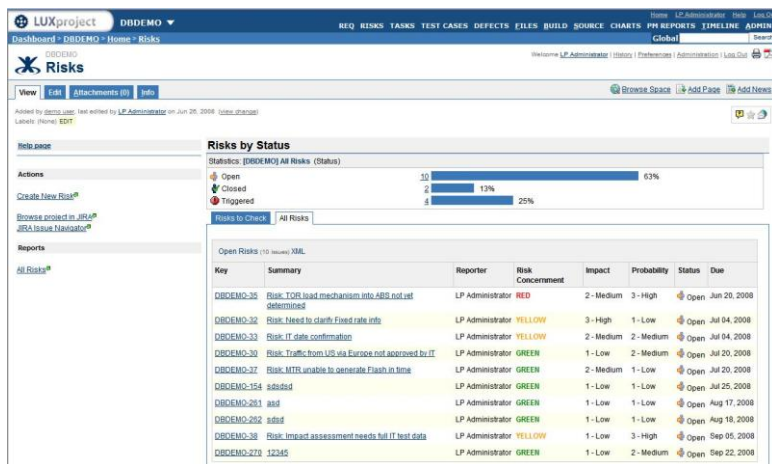


Рис 1. Иллюстрация интеграции JIRA функционала с помощью Wiki-движка Confluence

#### 4. Функциональная интеграция приложений

Основной целью системы LUXproject является поддержка процессов разработки программного обеспечения. Для этого используются интегрированные приложения, выполняющие определенную роль в процессе. Каждое приложение по сути представляет собой полноценный продукт, которым можно пользоваться независимо, но их интеграция вместе дает значительно больший эффект практически для всех активностей процесса. Основные цели

функциональной интеграции – это единая точка администрирования всех приложений и отдельных проектов, мониторинг активностей, возможность отслеживать связи артефактов процесса и управлять активностями.

Как известно, интеграция нескольких приложений попарно рано или поздно приводит к проблемам слишком жестких зависимостей, поэтому эффективнее реализуется с использованием паттернов Enterprise Messaging и использования шины сообщений. В последнем случае, при возникновении события в одном из компонентов, он посылает в шину сообщение, на которое подписываются заинтересованные стороны (другие компоненты), что позволяет достаточно быстро расширять функциональность всей системы, не затрагивая другие модули.

Кроме этого, вводится единый язык доменной модели, которым оперируют все слои интеграции отдельных компонент, преобразуя свои объекты в объекты единого языка и наоборот. Например, проектные роли эффективно отображаются на глобальные роли в JIRA и на группы в Confluence и на соответствующие сущности в других компонентах

Язык доменной модели представляет собой основные обобщенные сущности модели процесса разработки: роль, активность, артефакт.

На эту модель накладывается множество типизированных связей и ограничений, представленных в виде правил. Такая модель позволяет отслеживать связи между артефактами, поставляемыми разными компонентами (например, требование в wiki, test case, defect в JIRA и изменения исходного кода в репозитории Subversion), отслеживать целостность этих связей при помощи правил (например, запрещать вносить изменения в репозиторий, без указания дефекта в JIRA в соответствующем статусе).

Используемая модель и достаточно богатые функциональные возможности компонентов позволяют гибко настраивать систему под нужды конкретных заказчиков при внедрении.

## **5. Внедрение интегрированной системы**

Внедрение интегрированной системы имеет некоторую специфику. В связи с тем, что зачастую, в различных проектах компании могут применяться различные методологии разработки ПО. Простая инсталляция системы, с заложенной в нее типовой процессной моделью, не эффективна, так как произойдет несовпадение схем работы в системе и в жизни. Поэтому здесь нам видятся два подхода. Компания должна сама для себя ответить вначале на вопрос, насколько существующие процессы разработки ПО у нее эффективны (например, с помощью проектных аудитов).

Если компания пришла к выводу, что ее процессы не достаточно эффективны, то она может произвести «реинжиниринг» процессов путем принятия процессной модели, заложенной в интегрированную систему. В дальнейшем, сотрудники проходят обучение по новым схемам процессов работы.

Если компания считает, что ее процессная среда достаточно совершенна и накоплен существенный багаж знаний и опыт по проектам (особенно это актуально если используется системы, подобные JIRA), то в начале внедрения проводится сбор требований для уточнения схем процессов, статусов, форм отчетности. В дальнейшем, внедряемая интеграционная система настраивается под требования заказчика, и вводится в эксплуатацию.

Однако может так оказаться, что в центре разработок или компании, применяется несколько методологий разработки программного обеспечения. Одни проекты идут по практикам Agile, другие используют RUP процессы. Как быть в данной ситуации?

В компании LUXOFT, в рамках системы LUXproject, данную проблему решили путем создания проектных шаблонов. Проектный шаблон – это совокупность настроек функциональных модулей системы, (состоящих из issues, wiki-контента, версионного репозитория, шаблонов документов и базы знаний) под конкретную методологию разработки. На рис. 2. представлен пример визуального интерфейса проектного шаблона для Agile проектов.

## **6. Заключение**

Подводя итоги, можно сделать вывод, что описанная архитектура системы и схемы ее внедрения позволяют реализовать достаточно гибкий инструмент для управления процессами разработки ПО. Подход к интеграции компонентов системы, позволяет проектной команде управлять проектной средой

заточенной под конкретную методологию разработки, а также производить оптимизацию процессов уже во время выполнения проекта.

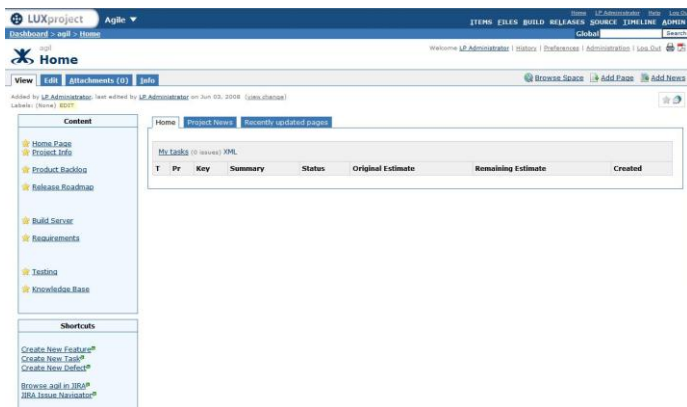


Рис. 2. Иллюстрация визуального интерфейса проектного шаблона “Agile”



# Automated Software Development Process Audit

Vadim Savkin

CQG

email: vadim@cqg.com

## Abstract

Internal software development process audit is a necessary instrument for ensuring high quality and stability of the process in software development companies. Goals of such audit are: a) ensure process compliance; b) assess software development process efficiency. Such audit is a rather expensive procedure. When it's automated it becomes much cheaper and gives possibility to run the procedure more frequently that leads to higher process quality and as a result higher software product quality.

Automated process audit is possible only if all valuable aspects of the process that need to be analyzed and controlled are collected in an integrated database with consistent way. Collected data must reflect real software development process with needed accuracy and details. Data collection procedures must guarantee correctness and consistency of the collected data. Experience shows that about 90% of the audit checks and more can be automated if data coverage for the process is good enough.

**Keywords:** internal process audit; metrics; dashboard; process data collection.

# **Автоматизированный контроль процесса разработки ПО**

**Вадим Савкин**  
**CQG**  
**email: vadim@cqg.com**

## **Тезисы**

Контроль процесса разработки ПО является необходимым инструментом поддержания высокого качества и стабильности процесса разработки ПО в компании. Целями такого контроля являются: а) обеспечение соблюдения заданного процесса разработки; б) оценка эффективности процесса разработки. Такой контроль является довольно дорогостоящей процедурой. Её автоматизация позволяет значительно снизить стоимость таких процедур, а значит, даёт возможность проводить их чаще, что способствует более высокому качеству процесса разработки ПО и, как результат, более высокому качеству разрабатываемых продуктов.

Автоматизация контроля процесса возможна при условии, что данные обо всех значимых аспектах процесса разработки, которые необходимо анализировать и контролировать, согласованно собираются и накапливаются в единой базе данных. Накапливаемые данные должны с необходимой точностью и детализацией отражать реальный процесс разработки. Процесс сбора данных должен гарантировать их полноту, корректность и согласованность. Опыт показывает, что около 90% проверок и больше при контроле процесса могут быть автоматизированы при должном покрытии процесса собираемыми данными.

**Keywords:** контроль процесса разработки; метрики; инструментальная панель; сбор данных о процессе.

## **1. Необходимость контроля процесса**

Средние и крупные компании-разработчики ПО сталкиваются с необходимостью контроля своего процесса разработки, если они хотят видеть этот процесс стабильным и

предсказуемым, гарантирующим стабильный высокий уровень качества разрабатываемых продуктов. Также такой контроль необходим, если компании хотят усовершенствовать свой процесс разработки. В этом случае контроль процесса позволит найти «узкие места» и оценить эффективность изменений.

Контроль процесса можно условно разбить на следующие части:

- Контроль соблюдения заданного процесса.
  - Проверка согласованности, полноты и корректности данных, собираемых различными системами поддержки разработки;
  - Проверка соблюдения стандартов процесса разработки, принятого в компании.
- Контроль эффективности и качества процесса.
  - Оценка качества разрабатываемых продуктов;
  - Оценка продуктивности разработки;
  - Оценка точности планирования;
  - Интервьюирование разработчиков и менеджеров относительно процесса.
- Принятие решений.
  - Принятие проектных решений по результатам проверок (в случае проектных аудитов);
  - Принятие решений о необходимости внесения изменений в процесс разработки.

Как правило, проверки и оценки при таком контроле проводятся в виде формальных процедур с использованием соответствующих контрольных таблиц (checklists). Результатом их является отчёт, содержащий следующую информацию:

- Список найденных замечаний и обобщённая оценка согласованности, полноты и корректности собираемых данных;
- Список найденных замечаний и обобщённая оценка соблюдения стандартов процесса разработки, сгруппированных по областям процесса;
- Список найденных замечаний по качеству разрабатываемых продуктов и обобщённая оценка их качества;

- Обобщённая оценка продуктивности разработки;
- Обобщённая оценка точности планирования.

Результаты таких отчётов обсуждаются на собраниях, делаются выводы о качестве и эффективности процесса, и принимаются решения о более тщательном контроле процесса (в случае его несоблюдения), о модернизации процесса (в случае обнаружения недостатков), о локальных изменениях процесса для конкретных ситуаций (если оказалось, что стандартный процесс не совсем удобен), о корректировке искажённых данных (если найдены замечания по корректности и согласованности данных).

Проведение вышеупомянутых проверок и подготовка отчётов может быть довольно длительной и дорогостоящей процедурой. Заметно упростить эту процедуру позволяет использование различных автоматизированных метрик процесса разработки, которые являются ключевым источником объективной информации о процессе. Более того, автоматизировать можно не только сбор необходимых метрик, но и сами такие проверки, что позволило бы существенно снизить стоимость контроля процесса разработки.

## **2. Инфраструктура для автоматизации контроля процесса**

В первую очередь, инфраструктура, обеспечивающая возможность автоматизации контроля процесса, должна позволять согласованно собирать и накапливать данные обо всех значимых аспектах процесса разработки, которые необходимо анализировать и контролировать. Все эти данные могут собираться с помощью разных программных средств поддержки процесса разработки ПО, таких как ALM-системы (Application Lifecycle Management), системы bug-tracking, task/time-tracking, системы планирования, системы контроля версий кода и т.п. Кроме того, процесс в компании должен включать в себя обязательное согласованное отслеживание (tracking) и измерение всех значимых аспектов процесса разработки в этих программных средствах. Все собираемые

данные должны с необходимой точностью и детализацией отражать реальный процесс разработки.

Говоря об измерениях процесса, следует назвать 3 базовых измеряемых величины процесса разработки ПО:

- Время (затрачиваемое на разработку);
- Размер (создаваемых артефактов);
- Количество дефектов (в артефактах).

На основе вышеперечисленных величин можно построить множество производных метрик для разных целей, которые будут использоваться при анализе качества процесса.

Инфраструктура для автоматизации контроля процесса должна приблизительно иметь вид, изображённый на рис. 1.

Центральным звеном такой инфраструктуры является единая база данных, содержащая в себе всю необходимую информацию о процессе разработки. Более-менее полную картину процесса можно получить, если в базе данных будет содержаться следующая информация:

- Иерархия продуктов и проектов с разбивкой на задачи.
- Иерархия департамента разработки ПО с данными обо всех разработчиках и менеджерах.
- Данные о времени, потраченном разными сотрудниками на ту или иную активность или задачу.
- Данные о созданных артефактах (требования, дизайн, тест-планы, код, проектная документация и т.п.) с подсчитанными размерами, причём должны храниться три значения для размера – добавленное, модифицированное, удалённое.
- Данные о найденных дефектах со всеми необходимыми атрибутами и историей изменений.
- Данные об инспекциях (в случае применения процесса инспекций), прочие данные, специфичные для процесса разработки, принятого в компании.
- Вспомогательные данные.

Наличие всех этих данных создаёт широкое пространство для анализа эффективности процесса разработки, поиска скрытых зависимостей, влияющих на качество и продуктивность разработки.

### **3. Корректность данных**

Необходимым условием использования данных и вычисления метрик на их основе является корректность этих данных, иначе они будут бесполезными. Процесс сбора данных должен гарантировать их корректность. Метрики вычисляются на основе «сырых» данных, которые могут собираться как автоматически, так и вручную. На корректность (т.е. согласованность и полноту) ручных данных влияет человеческий фактор, поэтому в них может таиться источник искажения метрик. Для решения этой проблемы можно использовать следующие приёмы:

- Возложение ответственности на исполнителей за корректность данных, относящихся к их работе (если допустил ввод некорректных данных, то обязан исправить).
- Периодические проверки согласованности данных и соблюдения стандартов независимыми экспертами, входящими в группу контроля процесса.

И всё равно даже при жёстком контроле корректности данных по разным причинам могут возникать локальные статистические выбросы, которые следует исключать при сборе статистических метрик.

### **4. Автоматизация контроля корректности данных**

Итак, корректность данных необходимо регулярно контролировать, иначе упадёт их достоверность. В этом могут помочь автоматизированные средства. Можно выделить 3 способа автоматизации контроля корректности данных:

- Контроль полноты данных в момент ввода. Такой контроль производится непосредственно в самих информационных системах, в которые эти данные вводятся, в момент ввода данных, т.е. система просто не позволяет завершить операцию, если данные некорректны. Это самый эффективный способ. Главный его недостаток – его

применимость сильно ограничена из-за ограниченности контекста отдельных операций ввода данных.

- Контроль с помощью автоматических уведомлений. Специальные программные агенты могут периодически или при наступлении определённых событий проверять корректность данных и при обнаружении несогласованности посылать специальные письма-уведомления людям, ответственным за эти данные.
- Периодические проверки с помощью автоматизированных средств при участии независимых экспертов (раз в неделю или несколько недель). Это более высокоуровневые проверки, которые нецелесообразно реализовывать с помощью вышеописанных способов. Результатом такой проверки будет список найденных замечаний, сгенерированный автоматически, и, возможно, расширенный замечаниями, найденными вручную экспертами.

Проверка полноты, согласованности и корректности данных является одной из задач контроля процесса разработки.

## **5. Процесс контроля и улучшения процесса**

Процесс контроля и улучшения процесса разработки ПО (рис. 2) с использованием автоматизации состоит из следующих основных шагов:

- Проверка соблюдения процесса:
  - Автоматизированная проверка собранных данных на полноту, корректность и согласованность.
  - Исправление найденных проблем в данных. Возврат на предыдущий шаг, если были проблемы.
  - Автоматизированная проверка соблюдения стандартов процесса разработки.
- Проверка эффективности процесса:
  - Автоматизированная оценка качества разрабатываемых продуктов, продуктивности разработки, точности планирования;
- Обсуждение результатов проверок на собрании:

- Просмотр и обсуждение найденных замечаний.
- Обсуждение эффективности отдельных аспектов процесса разработки.
- Принятие решений по улучшению процесса, а также проектных решений в случае проектных аудитов.
- Выполнение принятых решений.

## **6. Автоматизация проектных аудитов**

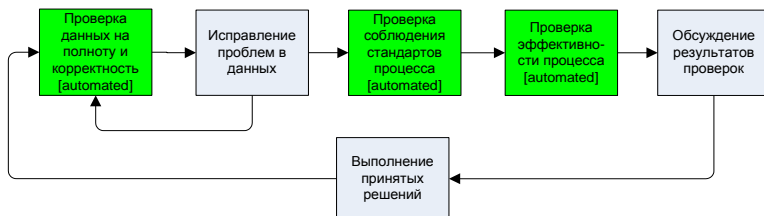
Формальный процесс аудита проектов включает с себя проверку различных показателей проекта, выявление тенденций, проверку соблюдения стандартов процессов и т.д. Набор всевозможных проверок составляется в контрольную таблицу (checklist), которая потом используется. Отдельные проверки в контрольной таблице следует описывать в формальном виде. Возможные шаблоны для таких проверок следующие:

- «должны быть введены такие-то данные» (проверка полноты данных или соблюдения стандартов процесса);
- «такие-то данные должны соответствовать тем-то» (проверка согласованности данных или соблюдения стандартов процесса);
- «параметр или метрика  $X$  в таких-то условиях должна находиться в пределах от  $X_{\min}$  до  $X_{\max}$ » - проверка на некий диапазон значений, который считается приемлемым и вычисляется на основе исторических данных (проверка соблюдения стандартов процесса или эффективности процесса).

Примеры проверок из практики компании CQG:

- «Каждая активность типа «разработка требований», «проектирование», «разработка тест-плана», «кодирование» приводит к созданию артефактов соответствующего типа с задержкой не более 2-х недель» - пример проверки на полноту и согласованность данных.





**Рисунок 2. Процесс контроля и улучшения процесса разработки ПО. Автоматизированные шаги выделены подстрокой [automated].**

- «Каждое изменение кода размером, превышающим 20 строк, проходит через формальный процесс инспекций» - пример проверки на соблюдение стандартов процесса.
- «Количество дефектов, найденных в ходе системного тестирования, меньше количества дефектов, найдённых в ходе интеграционного тестирования, которое, в свою очередь, меньше количества дефектов, найденных в ходе инспекций» - пример проверки качества процесса.

Подобные проверки могут быть легко автоматизированы при наличии необходимой информации в базе данных.

Каждый проект имеет свою специфику, поэтому часто бывает, что стандартные проверки могут плохо подходить для конкретного проекта. Для таких случаев будет удобно наличие возможности подстройки параметров автоматизированных проверок в зависимости от особенностей проекта.

Чем больше проверок удастся автоматизировать, тем дешевле получится стоимость проведения аудита. Однако абсолютно всё автоматизировать не получится, поэтому без «ручного» экспертного анализа не обойтись, но пробелы в автоматизации следует максимально сокращать. Опыт показывает, что около 90% проверок при контроле процесса могут быть автоматизированы при должном покрытии процесса собираемыми данными.

## **7. Автоматизация контроля процесса разработки в компании CQG**

Компания CQG является поставщиком данных и сервисов для биржевой торговли на основе разрабатываемых в компании программных систем. Штат департамента разработки насчитывает порядка 300 сотрудников, распределённых между 6 офисами в разных странах.

Разработчики ежедневно используют несколько средств автоматизации процесса разработки ПО, в которых параллельно с выполнением основных операций собираются данные о выполняемой работе:

- Система учёта рабочего времени и размеров произведённых артефактов с привязкой к конкретным проектам (собственная разработка).
  - Хранит иерархию департамента разработки ПО с данными обо всех разработчиках, менеджерах, и командах, а также иерархию продуктов и проектов.
  - В конце каждого рабочего дня каждый разработчик отмечает в этой системе, сколько времени на тот или иной проект и на ту или иную активность он потратил. Также разработчик вручную отмечает размер разработанных в этот день артефактов – требований, дизайна, тест-планов, документации.
  - Данные о размере разработанного кода попадают в эту систему автоматически. Также для подмножества проектов автоматически попадают данные о размерах требований и дизайна. Разработчику остаётся только привязать их к конкретному проекту.
- Система учёта отдельных задач и поддержки инспекций (собственная разработка).
  - В этой системе разработчики планируют и регистрируют свою работу на уровне отдельных небольших задач. Система полностью поддерживает методологию Personal Software Process (PSP).
  - Также эта система используется при проведении инспекций разрабатываемых артефактов. Сюда

записываются данные о найденных в процессе инспекций замечаниях и потраченном на инспекции времени (с точностью до минут).

- Система bug-tracking и репозиторий требований (разработка на базе Siebel).
  - Сюда записываются данные о найденных дефектах с историей изменений и предложения по улучшению программных систем.
  - Также в эту систему заносятся все разработанные требования.
- Система контроля версий (CVS)
  - При внесении изменений в код количество добавленных, изменённых и удалённых вручную строк кода подсчитываются специальным скриптом и заносятся в первую систему. Автоматически сгенерированный, а также перенесённый из одной версии продукта в другую, код не учитывается при подсчёте.

Все эти системы частично интегрированы друг с другом. Раз в сутки все данные о процессе из этих систем импортируются в единую базу данных, которая используется для сбора метрик и анализа процесса. В будущем планируется внедрение единой интегрированной системы взамен четырёх вышеперечисленных.

Для целей анализа собранных данных о процессе в компании CQG были разработаны специальные программные средства (на платформе MS Excel), объединяющие в себе функциональность визуальных инструментальных панелей (dashboards) и автоматизированных анализаторов метрик. Разработано несколько типов таких средств, отличающихся областью применения: Team Dashboard, Project Dashboard, Team-Project Dashboard, System Dashboard, Maintenance Dashboard. Эти панели-анализаторы используют информацию из единой базы данных. Результатами работы таких панелей-анализаторов являются:

- Набор графических диаграмм, показывающих изменение во времени (с недельными периодами) основных показателей и метрик процесса.
- Набор текущих числовых значений основных метрик.
- Набор детализированных данных из БД, относящихся к выбранному подмножеству.
- Список замечаний, найденных при применении формальных контрольных таблиц (checklists).
- Обобщённая оценка соответствия процесса стандарту на основе автоматических проверок.

Можно выделить следующие основные метрики, отображаемые в инструментальных панелях:

- Метрики производительности:
  - Скорость кодирования (число строк кода в неделю, считая чистое время кодирования).
  - Производительность (число строк кода в неделю, считая всё время, затраченное на любые активности)
- Метрики качества:
  - Плотность дефектов разных типов (число дефектов на 1000 строк кода).
  - Объём переделок (в процентах от общего объема работ).
- Метрики формального процесса инспекций:
  - Скорость просмотра кода (число строк в час).
  - Плотность найденных замечаний (число замечаний на 1000 строк кода).
  - Процент дефектов от общего числа замечаний.
- Метрики точности планирования:
  - Отклонение реальных значений затрат от запланированных (в процентах).
- Себестоимость проекта (\$).

В компании CQG автоматизированный контроль процесса разработки применяется на нескольких уровнях:

- Периодический аудит проектов (раз в 4-6 недель).
- Периодический контроль стандартного процесса разработки в командах (раз в 2-3 недели).

- Периодический контроль процесса поддержки (maintenance) продуктов (в зависимости от частоты релизов).

Контроль проводится независимыми экспертами, которые совмещают роль разработчика (в других подразделениях) с ролью инженера по процессам разработки ПО. При контроле используются автоматизированные панели-анализаторы, упомянутые выше. Помимо замечаний, найденных автоматически, эксперты могут выявить вручную и добавить свои собственные замечания. Далее, в зависимости от категорий найденных замечаний, они либо сразу устраняются (например, некорректность или неполнота данных), либо обсуждаются на последующем собрании.

## **8. Заключение**

Как известно, недостаточно задекларировать правила и стандарты процесса разработки. Чтобы он строго соблюдался и был эффективен, необходим его постоянный контроль. Автоматизация такого контроля возможна только в случае, если данные о процессе постоянно и согласованно собираются. Это в свою очередь накладывает дополнительные требования на процесс, обязывающие всех исполнителей аккуратно вводить данные о своей деятельности. По опыту CQG среднее суммарное время, которое тратит разработчик на работу со всеми программными инструментами сбора данных, составляет около 15 минут в день. Корректность и согласованность собираемых данных должна регулярно контролироваться. Но эти накладные расходы окупаются возможностью недорогого контроля соблюдения процесса разработки и анализа его эффективности, что, в свою очередь, приводит к более высокому и стабильному качеству разрабатываемых продуктов.

# Методы Программной Инженерии в Индустрии Компьютерных Игр

Елена Анатольевна Павлова  
Московский инженерно-  
физический институт  
(государственный университет)  
email: Elena.pav@gmail.com

Александр Викторович Гаврилов  
ООО «Майкрософт Рус»  
email:  
Alexander.Gavrilov@microsoft.com

## Тезисы

В настоящей работе рассматривается возможность применения методов программной инженерии в индустрии компьютерных игр. Анализируются особенности разработки игры как программной системы. Демонстрируется эффективность применения методов программной инженерии в задачах разработки игр на примере применения формальных предметно-ориентированных языков для разработки правил игр в классе пошаговых стратегий. Применение таких языков является новым подходом к разработке правил компьютерных игр.

**Ключевые слова:** компьютерные игры; методы программной инженерии; формальный язык; предметно-ориентированный язык.

## Введение

Компьютерные игры – одна из самых динамично развивающихся областей информационных технологий [1]. Доходы мировой индустрии игр за 2007 год возросли по различным оценкам на 60-67% [2, 3]. При этом, согласно данным Forbes и Games Industry [2], доход Европейского рынка видеоигр в 2007 году составил 17,9 миллиардов долларов, а рынка игр США – 18.8 миллиардов, что сопоставимо с доходами киноиндустрии. Число потребителей компьютерных игр также продолжает расти. Например, в Китае за 2006 и 2007 годы общее число игроков возросло на 6 миллионов человек и составило 40 миллионов [4]. Ожидания потребителей игр постоянно растут, порождая всё более высокие требования к качеству игр и разнообразию их содержания. Заметим, что кроме развлекательного аспекта, игры применяются в обучении и при создании систем подготовки различных специалистов [5].

Исторически индустрия игр являлась частью индустрии развлечений [1] и развивалась независимо от индустрии разработки программного

обеспечения. В результате, индустрия игр отличается значительно более развитой маркетинговой стороной по сравнению с индустрией программного обеспечения и значительно уступает по степени развития процессов разработки, применяемых методов, инструментальных средств, и, соответственно, отличается более низким качеством продуктов.

Часто разработка игр ведётся без использования устоявшихся методов разработки, процессы и методы изобретаются компаниями или отдельными командами самостоятельно «с нуля» для каждого проекта и являются закрытой информацией [6]. Документация может отсутствовать на протяжении нескольких этапов разработки и иметь слабоструктурированную форму, что отражается на её актуальности и приводит к противоречиям.

Например, необходимость проектной документации до сих пор вызывает оживлённую полемику и рассматривается значительной частью разработчиков как нововведение, допустимое для отдельных развитых компаний [7].

В литературе наиболее полно освещаются вопросы физического уровня проектирования игр и этапа реализации (примером могут служить работы [1, 8, 9, 10]). Большая часть работ посвящена решению задач трёхмерного моделирования и моделирования физических законов в играх (примером могут служить работы [8, 9, 10]). При этом общим вопросам разработки уделяется недостаточное внимание.

Отсутствие общепринятых методов разработки и общего научного обоснования является одной из причин низкой доли успешно завершённых проектов разработки компьютерных игр и недостаточно высокого качества выпущенных игр. Ежегодно в мире создается от 3 до 5 тысяч компьютерных игр, из них только несколько десятков становятся программными продуктами и появляются в продаже [11].

Вышесказанное определяет целесообразность применения методов программной инженерии в индустрии игр как способа повышения качества игр.

Задачей настоящей работы является рассмотрение методов программной инженерии и возможностей их адаптации с точки зрения применения в индустрии игр. Конкретно рассматривается возможность применения формальных предметно-ориентированных языков для разработки правил игр в классе пошаговых стратегий.

## **1. Специфика применения методов программной инженерии в разработке игр**

Вопросы применения методов программной инженерии к разработке игр рассматриваются, например, в работах [12, 13, 14] с точки зрения разработчиков и в работах [15, 16, 17] с научной точки зрения. Однако в перечисленных работах понятие разработки игр ограничивается любительскими проектами для небольшой команды разработчиков. Соответственно, рассматриваются лишь частные случаи применения методов программной инженерии в разработке игр, и не рассматривается влияние, которое может оказать применение методов программной инженерии на индустрию игр. Кроме того, в ряде случаев не учитывается специфика игр, рассмотрение которой с точки зрения программной инженерии позволило бы адаптировать ряд методов и, в итоге, повысить качество игры.

Рассмотрим подробнее особенности игры как программной системы. Специфика требований к играм состоит в следующем:

1. заказчиком игры, как правило, является её издатель. Цель заказчика состоит в получении максимальной выгоды от продаж игры, что отличает его от заказчика в индустрии программного обеспечения, где например, заказчик может планировать внедрение программы в своей компании и может быть явно заинтересован в функциональности программы
  2. аудитория игры, как правило, очень широка (с точки зрения возраста и рода деятельности), что порождает дополнительные трудности при сборе требований
  3. в игровой индустрии очень сильны творческий и инициативный аспекты, что часто усложняет сбор требований
  4. игры очень быстро устаревают морально, поэтому требования к срокам разработки игр выше, чем в индустрии программного обеспечения
  5. игры наиболее часто носят развлекательный характер и не предназначены для решения производственных задач. Одним из требований к играм является минимизация времени обучения игре.
  6. игры предъявляют очень высокие требования к аппаратному обеспечению, что объясняется, в основном, особенностями графических компонент. Например, игры часто требуют использования наиболее совершенных видеоадаптеров, аудиоадаптеров, наиболее быстрых шин передачи данных
  7. игры часто требуют использования устройств, которые редко используются другими классами программ (например, различные манипуляторы, рули, педали, сенсорные ковры)
  8. к некоторым частям кода игр предъявляются очень высокие требования производительности, например, к коду графических подсистем.
- Особенности проектирования игр состоят в следующем:



1. сценарии игр значительно отличаются от сценариев, создаваемых для вариантов использования в индустрии программного обеспечения, и более близки к сценариям кинофильмов
2. написание сценария игры и разработка концепции осуществляется нетехническим специалистом
3. значительную часть игровых проектов по сравнению с проектами индустрии программного обеспечения составляют работы дизайнеров, художников, музыкантов, специалистов по видео и трёхмерной графике. Процессы и методы для этой части проекта существенно отличаются от индустрии программного обеспечения.
4. циклическая разработка для художественной части игрового проекта считается слишком затратной и модель работы напоминает водопадную, что отражается на общей модели разработки игры и качестве игры
5. в индустрии игр существует общепринятое деление игры на высокоуровневые компоненты [1, 8]: логический, физический, графический компоненты, компоненты искусственного интеллекта, данных и взаимодействия с пользователем
6. некоторые аспекты проектирования и реализации игр достаточно хорошо проработаны в индустрии и нашли своё воплощение в компонентах, называемых игровыми движками. Несколько таких компонент используется при разработке практически каждой игры, так как реализация всех высокоуровневых компонент игры с нуля считается затратной и нецелесообразной. Использование сторонних высокоуровневых компонент накладывает значительные ограничения на проектирование и реализацию игры.

Особенности реализации игр состоят в следующем:

1. практически в каждом проекте используется сложная с точки зрения индустрии программного обеспечения графика. При этом доля проектов, использующих трёхмерную графику, продолжает расти.
2. значительная часть игр предполагает взаимодействие двух и более (до нескольких миллионов) игроков по сети в режиме «мягкого» реального времени.
3. значительная часть игр использует элементы искусственного интеллекта для моделирования поведения персонажей в игре.
4. в играх часто применяются системы захвата движения и сопутствующие программные модули

Следует отметить, что ряд аспектов применения методов программной инженерии в разработке игр уже достаточно хорошо проработан. Так, сопровождение и управление конфигурацией в ряде случаев уже вышли на промышленный уровень [18]. При этом процессы сбора требований, проектирования и конструирования программного обеспечения зачастую остаются на ремесленном уровне «штучных поделок» [1, 8].

Недостаточно развиты области тестирования и управления качеством. Например, тестирование во многих случаях выполняется вручную разработчиками до выпуска продукта или перекладывается на плечи пользователей после выпуска. Этот принцип получил широкое распространение и описывается в литературе как “play a little, tweak a little” [1, 8] (немного поиграй и настрой).

Масштабное использование методов программной инженерии (в особенности в проектировании и конструировании) может позволить увеличить долю успешных проектов, повысить общий уровень качества продуктов и снизить себестоимость создания программного продукта за счет создания специализированных средств разработки и повторного использования компонент и шаблонов.

## **2. Применение формальных методов программной инженерии для проектирования правил игр в классе пошаговых стратегий**

В качестве примера применения методов программной инженерии в разработке игр рассмотрим разработку игр в классе пошаговых стратегий, специфичные для этого класса задачи и решение одной из задач при помощи формальных методов программной инженерии. Согласно работе [19] формальные методы (в частности формальные языки), относятся к области знаний (SE knowledge area) методов и средств программной инженерии (software engineering tools and methods).

Варианты классификации игр и особенности пошаговых стратегий детально рассматриваются в [1, 8, 9, 10]. В качестве исследуемой задачи рассмотрим разработку правил игры, как одну из ключевых для выбранного класса.

Разработка правил игры включает в себя этап проектирования правил [1], которому с точки зрения программной инженерии могут быть поставлены в соответствие сбор требований и создание концепции. Затем следует этап проверки и балансировки правил. Далее выполняется реализация правил в виде кода на языках программирования высокого уровня. Этот этап может быть отнесён к этапу реализации в терминологии программной инженерии. Как правило, дизайнер периодически (десяtkи раз) вносит исправления в правила и процесс повторяется с начала [1]. Заметим, что в описанном процессе с точки зрения программной инженерии полностью отсутствует значительная часть этапа проектирования.

Проектирование правил игры традиционно выполняется вручную дизайнером игр [1, 8], который не является техническим специалистом [1]. Правила представляются в виде слабо структурированных документов и обширных таблиц. Автоматизация разработки правил, их последующей обработки и генерация кода на основе правил существенно затруднены. В настоящей работе в качестве одного из вариантов решения проблемы будет рассмотрена разработка и применение специализированного средства

создания правил, использующего свой внутренний формальный язык представления правил.

Такое специализированное средство позволит явно проектировать правила игры на концептуальном, логическом и физическом уровнях. Будет показано, что такое решение, характерное для задач автоматизации в индустрии разработки программного обеспечения, имеет ряд преимуществ и является предпочтительным по сравнению с альтернативными решениями.

### **3. Применение языка для проверки корректности правил игр**

В основу разработки инструментального средства положена концепция использования формального предметно-ориентированного языка для разработки правил игр в классе пошаговых стратегий. Правила игры включают множество типовых сценариев игры.

Предложенный предметно-ориентированный язык позволяет описывать сущности из предметной области конкретной игры (а именно связанные с сущностями данные и поведение), определять ограничения, налагаемые на поведение сущностей, определять отношения между сущностями, определять реакцию на поведение некоторой сущности со стороны других сущностей.

Корректность модели сценария игры проверяется путём проверки синтаксического и семантического соответствия интерфейсов взаимодействующих объектов модели при помощи метода, предложенного в работе [20].

Прототип результирующей программной системы получается на основании трансформационного подхода в рамках преобразования конструкций предметно-ориентированного языка на основании семантических правил.

Таким образом, использование языка позволяет не только описывать модель правил в терминах дизайнера игр, но и осуществлять быструю автоматизированную разработку верифицируемого прототипа.

### **Заключение**

В настоящей работе была рассмотрена возможность применения методов программной инженерии в индустрии компьютерных игр. Проведённый анализ индустрии показал актуальность задачи повышения качества игр за счёт применения методов программной инженерии в разработке игр. Анализ специфики процесса разработки игр позволил определить этапы разработки игр, для которых наиболее целесообразно применение указанных методов и их адаптаций.

В настоящей работе рассмотрено использование формального предметно-ориентированного языка для создания средства разработки правил игр в классе пошаговых стратегий. Предложенный подход на основании формального языка дает возможность разработать инструментальное

средство, позволяющее более качественно проектировать правила игры. На примере использования языка была продемонстрирована эффективность методов программной инженерии для решения задач разработки компьютерных игр.

## **Источники**

- [1] Rollings A., Morris D. Game Architecture and Design. A New Edition.– Indianapolis: New Riders Publishing, 2004
- [2] Androvich M. Forbes: Europe is least mature videogame market. – <http://www.gamesindustry.biz/articles/forbes-europe-is-least-mature-videogame-market>, 2008
- [3] Путинцев Т. Игровая индустрия растёт. – <http://www.dtf.ru/news/read.php?id=31442>, 2003
- [4] В Китае резко возросло число игроков. – <http://www.lenta.ru/news/2007/05/07/china/>
- [5] Бондарева Т.В., Грызлова О.В., Евтюхин Н.В. Компьютерные деловые игры как инновационное средство обучения /Телекоммуникации и информатизация образования, (2001), 1 (январь), 58-69
- [6] Плюммер Дж. Гибкая и масштабируемая архитектура для компьютерных игр. – <http://www.dtf.ru/articles/read.php?id=40757>, 2004
- [7] Материалы конференции разработчиков компьютерных игр. КРИ 2007, – [http://kricnf.ru/2007/index.php?type=info&doc=speech\\_records](http://kricnf.ru/2007/index.php?type=info&doc=speech_records), 2007
- [8] Wihlidal G. Game Engine Toolset Development.– Boston, MA: Thomson Course Technology PTR, 2006
- [9] Meigs T. Ultimate Game Design: Building Game Worlds. NY: McGraw-Hill/Osborne, 2003
- [10] Bates B. Game Design. Second Edition. – Boston, MA: Thomson Course Technology PTR, 2004
- [11] Консалтинговая группа MD. – <http://www.md-consulting.ru/>
- [12] Gruhl R. XNA Game Studio Express, an Overview. Proceedings of GDC 2007 – <https://www.cmpevents.com/sessions/GD/S4440i1.ppt> , 2007
- [13] Flynt J. P., Salem O. Software Engineering for Game Developers. – Boston, MA: Thomson Course Technology, 2004
- [14] Rucker R. Software Engineering and Computer Games.– London: Addison Wesley, 2003
- [15] Yang H-C. A General Framework for Automatically Creating Games for Learning. Proceedings of ICALT'05. – IEEE, 2005, p. 28-29

- [16] McNaughton M., Cutumisu M., Szafron D., Schaeffer J., Redford J., Parker D. Script-Ease: Generating Script-Code for Computer Role-Playing Games. Proceedings of ASE'04. – IEEE, 2004, p.386-387
- [17] Meng L.S., Prakash E. C., Loh P. K. K. Design and Development of a Peer-to-Peer Online Multiplayer Game Using DirectX and C#. – IEEE, 2004, p. 278-281
- [18] Common Questions about Blizzard – <http://www.blizzard.com/us/inblizz/genfaq.html>, 2008
- [19] Guide to the Software Engineering Body of Knowledge. 2004 Version. SWEBOK. – Los Alamitos, California: IEEE, 2004
- [20] Гаврилов А.В., Павлова Е. А. Формализация проектирования сложных информационных систем на основе анализа функциональных интерфейсов. Информационные технологии №9, 2008. – М.: Новые технологии, 2008, стр. 9-15

## **All in one: you can do more. We'll teach you how...**

**Alexander V. Babich**  
**Knowledge Center Incom company**  
**email: alexander.v.babich@acm.org**

### **Abstract**

The report presents some results of author's research in the area of professional training framework based on the results of detailed customer audit. Author will share his experience as well as the some further ideas.

Proposed approach consists of the detailed audit of customer's goals, technologies and business-processes. Based on audit's results we compile a set of competencies needed to achieve customer's goals. Next step is to select target groups to be trained, audit their current skill set, prepare educational and development plans...

The research was conducted in Knowledge Center Incom Company for two of our customers since January 2008. At this time we have great results: customer loyalty, massive sales, and new horizons for new researches...

**Keywords:** IT-education, Microsoft, Incom, Knowledge Center, audit, human development, education planning.

# **Все в одном: повышение конкурентоспособности клиентов через обучение**

**Александр Бабич**  
**Центр Знаний компании Инком**  
**email: alexander.v.babich@acm.org**

## **Тезисы**

Будут представлены некоторые результаты практических экспериментов автора в области продвижения образовательных услуг Центра Знаний, основываясь на результатах аудита компании-клиента.

Предложенный подход включает тщательный аудит целей компании, используемых технологий и бизнес-процессов, на основе результатов которого формируется набор компетенций, развитие которых позволит компании добиться поставленных целей. Затем выделяются целевые группы, для которых составляются планы обучения и развития.

Эксперимент проводится на базе Центра Знаний компании «Инком» для двух компаний-клиентов с начала текущего года. Полученные результаты позволили усовершенствовать методику, повысили лояльность вышеупомянутых клиентов, открыли новые горизонты для дальнейших исследований.

**Keywords:** ИТ-образование, Microsoft, Инком, Центр Знаний, аудит, развитие персонала, планирование обучения.

## **1. О чем пойдет речь**

Значение обучения и сертификации персонала для бизнеса компании сегодня понимают почти все отечественные работодатели. В связи с этим бурно расцветают всевозможные СТЕС'и, «компьютерные» курсы, зачастую подобные услуги предоставляют и образовательные учреждения.

Однако, мало кто задумывается над тем, что ИТ-обучение – это не просто продажа тренингов, а нечто большее. Привлечь и удержать клиента, превратить каждый впервые проведенный тренинг в начало долгого, плодотворного и взаимовыгодного сотрудничества – не так-то просто. Данная работа – попытка рассказать о том, как это делаем мы, а именно Центр Знаний компании Инком [1], пользуясь методикой комплексного подхода к продвижению образовательных услуг.

Но сначала давайте познакомимся немного ближе.

## **2. Кто есть кто на рынке образовательных услуг Украины**

Игроков на рынке образовательных услуг Украины – достаточно много, но столичный Центр Знаний заслуженно занимает лидирующее положение. Осмелюсь предположить, что и не только в Украине – даже огромная Россия может похвастаться всего лишь тремя учебными центрами похожего «калибра».

Основан Центр Знаний в 2002 году и за это время была проделана огромная работа, а главное – подобрана отличная команда профессионалов.

В данный момент наш учебный отдел состоит из 6 сертифицированных инструкторов CISCO, 5 сертифицированных инструкторов Microsoft, 2 инструкторов Sun/Linux, 1 инструктора Oracle и 1 инструктора по управлению проектами, общее число сертификационных статусов которых превышает 170.

На нашем счету - более 50 реализованных проектов в Казахстане, России, Польше, Словении, Армении, Грузии, Греции, США, Англии, Бразилии, Пакистане, ОАЭ, Африке и др, а также более 20 разработанных авторских тренингов.

На данный момент Центр Знаний - единственный в Украине имеет статус Cisco Learning Partner / CLSP.

Материальная база – тоже на высоте. В наличии 10 прекрасно оснащенных классов на 150 мест, а общая стоимость лабораторного оборудования превышает \$1000000. Также имеется загородный учебный центр с классом на 40 мест, гостиничными номерами со всеми удобствами, бассейном, сауной, бильярдной и т.д.

В своей работе Центр Знаний всегда в первую очередь ориентировался на среднего и крупного корпоративного заказчика. Именно поэтому в последнее время мы стали задумываться над тем, как сделать наше взаимодействие с клиентом более тесным и эффективным, превратить обучение персонала в многолетний проект, стали пытаться выработать методику, позволяющую предлагать именно те услуги, которые максимально соответствуют нуждам наших клиентов.



### **3. Наш подход: не просто продавать тренинги, а развивать клиента!**

В чем же состоит секрет нашей методики, которую мы уже весьма успешно пытаемся применять с начала текущего года?

Секрет прост – прежде чем продавать клиенту какие-то тренинги (будь то авторизованные или авторские курсы), мы проводим тщательное изучение компании, определяем, какие именно знания, умения и навыки необходимы персоналу для достижения целей, преследуемых клиентом.

На следующем этапе мы анализируем структуру компании и определяем целевые группы для обучения.

Следующий шаг – оценка потребностей персонала в развитии и, собственно, построение подробных планов обучения с учетом уже имеющихся знаний, умений и навыков, карьерных путей сотрудников и т.д.

А далее мы проводим обучение и повторяем все сначала, итеративно. Таким образом, мы добиваемся качественной подготовки персонала и помогаем клиенту спланировать развитие сотрудников на годы вперед, причем не просто развитие, а развитие с учетом целей, стоящих перед компанией, технологий, используемых клиентом и планируемых к внедрению, его бизнес-процессов и т.д.

Далее мы подробно рассмотрим каждый из этапов описанного выше процесса.

### **4. Чего хочет клиент от жизни?**

Итак, первый этап – это аудит компании-клиента. В общих чертах этот процесс иллюстрируется приведенной ниже схемой.

Цель данного этапа - сформировать набор компетенций (Skills Set), который позволит компании достигнуть намеченных целей.



Действительно, все очень просто – на основе предоставленной клиентом информации о бизнес-целях компании и ее планах, используемых и планируемых к внедрению технологиях, уровне зрелости инфраструктуры и бизнес-процессов, мы получаем список компетенций, которыми должны обладать сотрудники компании.

Какие средства мы для этого используем? Очень простые – анкету-опросник, содержащую вопросы, затрагивающие все перечисленные аспекты. Заполняется она руководством компании-клиента. А еще у нас есть список компетенций, «синхронизированный» с анкетой, так что, результатом ее анализа сразу же становится некий перечень стандартных компетенций, которые клиенту нужно развивать.

Причем, как мы уже упоминали, список этот составляется «на перспективу» - он включает в себя не только те компетенции, которые необходимы компании в данный момент, но и те, которые ей понадобятся в ближайшем будущем.

## **5. Что мешает компании достичь поставленных целей?**

А мешают компании двигаться вперед люди. Вернее, недостаток у них знаний, умений и навыков. И именно поэтому на втором этапе мы пытаемся определить, сотрудников каких именно департаментов и рабочих групп нужно развивать.

Идея довольно проста – путем сравнения организационной диаграммы компании-клиента с некоей шаблонной организационной диаграммой, мы «разбрасываем» полученный на предыдущем этапе список компетенций по подразделениям / группам / позициям и т.д.

Графически этот процесс можно представить довольно просто:



Опять-таки, инструменты довольно просты и логичны – эталонная организационная диаграмма, с каждым элементом которой сопоставлены компетенции из нашего списка. Таким образом, выделив список компетенций на первом этапе, мы сразу же знаем, кого нужно развивать. Осталось только идентифицировать подразделения / группы / позиции компании –клиента, соответствующие нашим «эталонным».

## **6. Дальше дело за тренерами? Еще нет.**

Кого именно нужно развивать, мы уже знаем. Возникает вопрос – как? То есть, цель третьего этапа – составить подробные планы обучения людей из целевых групп.

К тому же, планы эти должны учитывать карьерные пути сотрудников и знания, умения и навыки, которыми вероятно люди уже обладают.

С этой целью мы проводим оценку компетенций, уже имеющихся у сотрудников компании на данный момент, анализируем карьерные пути сотрудников, что позволяет нам спланировать развитие персонала на годы вперед.

Для этого у нас имеются анкеты и тесты для оценки имеющихся у персонала знаний, умений и навыков, а также схемы логической взаимосвязи тренингов, где каждой компетенции из нашего списка сопоставляется набор курсов, которые рекомендуется пройти, чтобы получить необходимые знания, умения и навыки.

Схематически это можно показать так:



## Планы обучения сотрудников

### 7. Что дальше?

А дальше, собственно, обучение персонала и следующая итерация - повтор всей описанной выше последовательности этапов.

При этом, естественно, автоматизировать процесс можно лишь частично, но даже в текущем варианте данная методика позволяет работать с клиентом более эффективно. Тем не менее, приоритетным направлением развития методики нам видится именно ее автоматизация.

При разработке данного подхода мы, конечно же, учитывали опыт, полученный нами в ходе работы над сертификацией SEP [2], материалы CMMi [3], ITIL [4], а также модель зрелости инфраструктуры Microsoft [5]. В процессе практического применения методики мы будем стараться итеративно улучшать ее, так что каждый новый клиент поможет нам сделать наш подход более эффективным.

### 8. Заключение

И наконец, внедрение. В январе этого года мы рискнули поэкспериментировать в вышеописанном стиле с двумя из наших крупных корпоративных клиентов.

Причем, особых доополнительных затрат на разработку инструментов такого анализа у нас не было – удовлетвоились созданием несложных InfoPath-форм.

И результаты превзошли все ожидания – применение методики позволило повысить продажи тренингов, поднялась лояльность наших клиентов, вырос их CS (уровень удовлетворенности), про мощнейший же маркетинговый эффект, думается, даже и говорить не стоит.

Конечно, описанный подход далеко не совершенен, но после некоторой доработки мы планируем начать применять его для всех наших клиентов. В конце-концов, выгоду от этого получают все...

## **9. References**

- [1] <http://edu.incom.ua>
- [2] <http://www.tekama.com/index.php?page=46>
- [3] <http://www.sei.com>
- [4] <http://www.ital-officialsite.com/home/home.asp>
- [5] <http://www.microsoft.com/rus/midsizebusiness/solutions/itinfrast ructure/stages/default.msp>

# Архитектура высокопроизводительных web-приложений.

Ковалевский Владимир Александрович  
Руководитель отдела разработки компании  
«Миланор» (WMJ.RU)  
email:ikik1@yandex.ru

## Abstract

In the modern world of internet and **web-applications** there are a lot of typical solutions for your brand new web-site, but still if you want to have a serious internet business with a great traffic load and huge page views counters, you have to collaborate with a software development team, if it is so and you've made the decision, text below is for you and for your team.

It is very important to get together three parts in order to draw a good and fast web-application. Here they are: simple and flexible architecture design, interactive cache system, high-performed database with not complicated queries. My idea is not of any original kind, these three points are spread into a complex project and experience of creating fast and interactive web-application (talking about 100.000 – 1.000.000 unique users generating up to 700.000 – 6.000.000 views, according to our stress tests<sup>4</sup>).

So first of all we've hit the design and choosing right level of abstraction was very important, because at this stage we have to decide how deep our database structure will be “denormalized”, and how much CPU time will be lost forming and counting on an inheritance, object mapping, abstraction and other “perfect” object oriented design features.

Next we have to draw a simple cache system with objects and queries support. It is very important to determine how to cache queries, how to store there an objects and of course how to organize links between objects and queries. OR mapping principals helped us to clarify the situation and had connected our “perfect” object model to a data storage.

---

<sup>4</sup> Web server – HP Proliant 3.2 Ghz (Dual Core), 4GB RAM; SQL Server HP Proliant 2.7 Ghz (Dual Core), 8 GB RAM.

The last step was to make fast database architecture. It was simple: well-organized automated archive and consolidated tables (that include data from many others) solved the problem.

The experience of our team gives a key how build a high-performed web system; design production, cache structure and data storage concepts are shown after this small abstract part.

**Keywords:** Web-application; Cache system; Design concept; High performance; Software development.

## Часть 1. Дизайн.

В данной части мы не рассматриваем конкретные примеры дизайна, а даём общие рекомендации и строим концепцию того, как проектировать систему так, чтобы она была быстрой, расширяемой и немного поддерживаемой.

### Уровни абстракции и основные шаблоны проектирования для производительных web - приложений.

Очевидно, что любое web-приложение в конечном итоге, независимо от клиентской и серверной частей является парой запрос-ответ, обмен которыми происходит по протоколу HTTP, в заголовке которого передаются некоторые переменные, а в теле данные. Т.е. в данном случае целесообразно ввести некоторое абстрактное понятие Контент (слово пришло из сферы деятельности контент-менеджеров) – то с чем работает наше приложение, отправляя запрос или получая ответ от сервера.

Два других понятия, которые мы введём, будут характеризовать позицию или координаты нашего Контента относительно всего объёма данных, с которым работает наше приложение – Рубрикатор и Группа. Эти понятия представляют собой две древовидные структуры данных. Таким образом, Контент всегда «располагается» в некотором Рубрикаторе и входит в абстрактную Группу (рисунок 1).

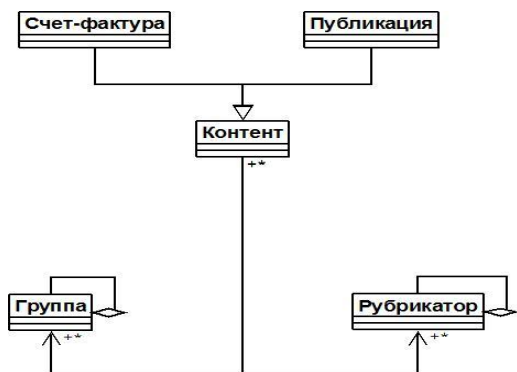


Рисунок 7 (Положение Контента)



Эти на первый взгляд общие или размытые понятия могут иметь явное практическое применение, например с их помощью мы можем осуществлять простую навигацию в нашем приложении:

[http://домен.ru/рубрикатор1/рубрикатор2/ключевоеслово/группа1/группа2/идентификатор\\_контента/параметры](http://домен.ru/рубрикатор1/рубрикатор2/ключевоеслово/группа1/группа2/идентификатор_контента/параметры).

Необходимость введения двух древовидных структур объясняется просто, на примерах:

- Фотография (Контент), находится в разделе Автомобиля (Рубрикатор) и входит в альбом Мой Ламборджини (Группа).
- Платёжное поручение (Контент), находится в Бухгалтерии (Рубрикатор) в офисе Тверская (Группа).

Можно обойтись одной структурой, но работать с одной структурой будет сложнее и медленнее, чем с двумя, так как увеличится количество уровней вложенности, а следовательно, процессорное время и не будет явного разделения сущностей (всё будет в одной куче). Можно вводить более двух структур, но поддержка и построение такой системы будет более громоздким и сложным.

Ключевым признаком производительности в данном подходе является определение типа связей: Контент-Группа, Контент-Рубрикатор, если связи определены, как многие-ко-многим – мы теряем в производительности из-за неоднозначности положения Контента, но получаем гибкость при определении различных типов Контента, например: публикация, платёжное поручение, видео, могут находиться в нескольких рубрикаторах, и возникает проблема выбора из какого именно Рубрикатора получать один и тот же Контент и скорее всего, как следствие, будет написан лишний код и станет ощутима потеря производительности системы в целом. Ограничивая себя связями один-ко-многим (Рубрикатор-Контент, Группа-Контент) мы выигрываем в производительности, но теряем гибкость системы в целом, вводя понятие уникальности Контента. Какой тип связей использовать выбор за вами.

Кроме определения координат Контента, нам хотелось бы его описать – введём Типизацию и будем использовать данное понятие, выражаясь языком разработчика, как мета информацию для наших объектов, наследников Контента.

Введём в нашу систему ещё одно понятие, важное для производительности – Статистика Контента. Часто для простых страниц и даже для сложных, нам не надо иметь дела с самим Контентом или его массивом, сам объект в том или ином случае использования, содержит массу ненужной в данный момент информации, требующей время со стороны процессоров наших серверов. Будем собирать необходимые нам данные (количество правильных ответов, анонсы, итог по расчетному периоду, количество комментариев, число сотрудников в компании) в некоторые статистические сущности, которые будут располагаться в системе по тем же принципам и законам, что и сам Контент. Таким образом, нам достаточно обратиться к Статистике Контента один раз и получить необходимый ответ, чем опрашивать всех участников нашего запроса (весь массив Контента).

Подводя итог, мы можем сказать, что построили для нашей «абстрактной команды» некую среду, в которой есть все необходимые для построения и развития архитектуры термины: Контент, координаты Контента (Рубрикатор, Группа), Типизация, Статистика Контента.

Хотелось бы отметить, что данная, достаточно общая концепция имеет конкретное практическое применение:

- Системы публикаций сайта.
- CMS системы.
- Платёжные сервисы.
- Интерактивные сервисы (игры, форумы, блоги, соц. сети) и т.д.

А также является залогом производительности веб-приложения, исключая большинство ошибок проектирования, которые могут сказаться на его быстродействии.

### **Проблемы безопасности.**

Всем прекрасно известны два метода: аутентификация и авторизация, для защиты от несанкционированного доступа, проверки подлинности и назначения ролей. Существует огромное количество модулей, встроенных в веб-серверы, обеспечивающих данный механизм. Мы сознательно не будем рассматривать данный вопрос с этой точки зрения, а обратимся к построению политик безопасности и разграничению доступа к Контенту с точки зрения сценариев его использования.

При проектировании систем мы часто сталкиваемся с проблемой повторяющихся типовых операций, накладывания одних и тех же фильтров, определения правил взаимодействия между объектами. В общем случае все эти действия можно определить, как политики, например:

- Показывать только актуальный Контент.
- Показывать скрытый, удалённый Контент.
- Показывать Контент, только его владельца.

В последнем примере мы видим взаимодействие политики и роли. Таким образом, говоря языком разработчика, мы можем сказать, что каждый объект наследник от Контента обладает теми или иными политиками безопасности и в общем случае данные политики зависят от Роли пользователя в системе.

На практике, такой подход отражается в создании определённого модуля, «Менеджера» - его основной функцией будет являться регистрация новых политик и применение к запросам уже зарегистрированных.

### **Системная архитектура глазами администратора**

Содержание данного раздела, будет носить скорее практический оттенок, с целью обосновать и подчеркнуть важность Администратора приложения в рамках проблем отказоустойчивости и быстродействия.

Для обеспечения быстродействия и стабильности нашего приложения на нескольких серверах нам понадобится Администратор, но не системный, а Администратор приложения.

Простой системный Администратор, мало, что может сделать при нестабильной работе приложения на больших нагрузках. Максимум, чего он может добиться – это логирование ошибок и наращивание мощности. В случае если такой специалист обладает знаниями об архитектуре приложения и, которому предоставлена возможность управлять его настройками, то он может обеспечить требуемые параметры простой оптимизацией архитектуры, настройками распределения памяти и процессорного времени. Например:

- Оптимизировать структуру рубрикатора.
- Организовать перенос неактуальных данных в архив.

- Настроить распределение памяти для разных модулей приложения.
- Управлять политиками безопасности.
- Определять уязвимые места и определять ошибки разработчиков и т.д.

Именно расширение круга интересов Администратора, даёт ему больше возможностей для того, чтобы обеспечивать отказоустойчивость и производительность системы.

Сформулируем для примера несколько общих, но крайне важных для нашей системы требований, которые могут быть предоставлены Администратором и включены в проект:

- Модульность – система (приложение) должно состоять из подсистем (модулей), таким образом, чтобы выход из строя одного из них не привёл к отказу системы в целом и нарушениям в работе ядра.
- Утечка ресурсов – при стабильной нагрузке и постоянном среднесуточном трафике, система (приложение) должна занимать неизменяемый или меньший объём оперативной памяти и процессорного времени.
- Диагностика – система (приложение) должна предоставлять набор средств, для диагностики и управления.

## **Часть 2. Кэширование.**

В данной части мы затронем вопрос о буферизации данных, рассмотрим различные уровни кэширования от хранилища данных до html вёрстки.

### **Три уровня кэширования.**

Представим, что мы построили некоторый образец нашей системы. Пусть это будет простая система публикаций – некий информационный портал. Отдадим наше приложение Администратору и получим его обратно с диагнозом – неудовлетворительные стресс тесты. Действительно, разбирая графики, мы видим полную загрузку CPU хранилища данных и «простой», практически не загруженный процессор серверов приложения.

Решение очевидно – используйте кэширование, но какое, как, и когда оно уместно, а когда нет?

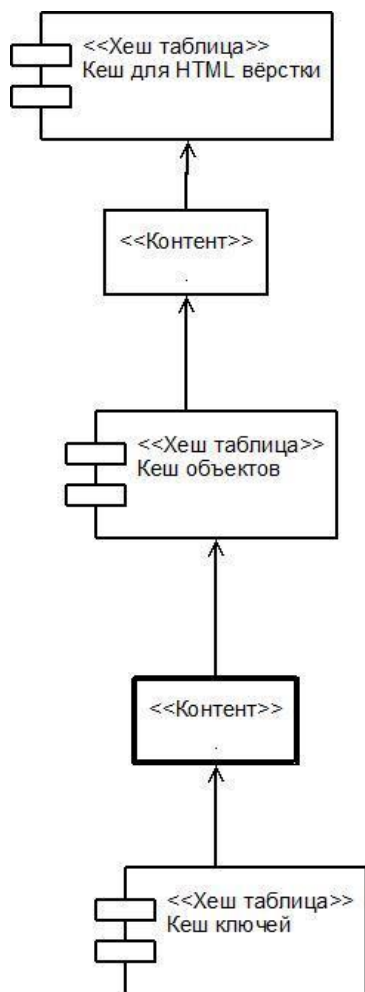
Для того чтобы понять и разобраться в данной проблеме, а главное предложить разработчикам адекватное решение, давайте мысленно разделим данные, которыми оперирует наше приложение, на две части: статичные (редко обновляемые) и интерактивные (требующие обновления в течение времени – «интерактива»). А с другой стороны введём категорию их представлений: близко к интерфейсу (часто запрашиваемые) и далеко от интерфейса (редко запрашиваемые).

Рассмотрим пересечение этих двух, вымышленных категорий:

- Статичные данные, близкие к интерфейсу (Новости на сайте).
- Статичные данные, далекие от интерфейса (Архив данных).
- Интерактив близкий к интерфейсу (Сервисы, форум, блог).
- Интерактив далёкий от интерфейса (Статистика, рейтинги).

Из представленного пересечения видно, что в некоторых случаях кэширование должно выполняться максимально близко к веб-серверу – кэшируется сама вёрстка (HTML код страницы), например Новости. В другом случае, кэшировать лучше объекты, там же изменять их. Или вовсе кэшировать только результаты (ключи запроса), а объекты каждый раз брать из хранилища. Мы видим, что сценариев может быть огромное количество, но в каждом случае мы используем тот или иной кэш: Кэш с вёрсткой, кэш объектов или кэш ключей.

Объект, говоря метафорами, совершает путешествие (Рисунок 2) в нашей системе из хранилища данных, через уровни кэша, к пользовательскому интерфейсу и обратно, тем самым распределяя нагрузку между серверами приложения и хранилищем данных.



**Рисунок 8 (Объект в кэше)**

### **Представление объекта в кэше.**

Предположим, что основная нагрузка нашего приложения придётся на кэш объектов, которые мы будем получать из хранилища по ключам запроса, который в свою очередь будет находиться в кэше ключей.

В общем случае кэш объектов может содержать любые объекты, любого типа и тогда вычислений, связанных с определением типа и приведением к типу не избежать, а также

использование объекта внутри ядра системы будет крайне затруднительным, мы попросту не знаем, с чем работаем.

Пусть кэш будет однородным, т.е. все объекты в кэше имплементируют некий общий интерфейс, тогда многих проблем, возможно, мы избежим, так как мы заранее договорились с определением взаимодействия объектов с кэшем и ядром.

Это простое, на мой взгляд, крайне важное правило существенно упрощает понимание системы и её проектирование, а также положительно влияет на наши стресс тесты.

### **Кэшируем запрос.**

Итак, как мы выяснили, все объекты находятся в кэше, и все они имплементируют некий интерфейс. Основная задача этого интерфейса – предоставить уникальный идентификатор объекта для однозначного определения его положения в кэше, таким образом, избегая избыточного сканирование всех объектов.

Пусть результатом нашего произвольного запроса к данным всегда будет некоторый набор ключей, по которому мы сможем получить наши объекты. Тогда, создавая кэш ключей, мы освобождаем систему от обращения к данным в том случае, если запросы одинаковые.

### **Кэшируем вёрстку.**

В некоторых случаях данные расположены настолько близко к интерфейсу и достаточно статичны, что у нас отпадает необходимость в наличии объекта. В таких случаях мы будем кэшировать результат http запроса или какую-то его часть.

Это реализовано на многих web-серверах, но если мы введём зависимость кэш вёрстки от кэша объектов, то теоретически мы сможем не контролировать более этот уровень кэширования. Объекты сами будут решать, кэшировать ответ сервера или нет.

Хотелось бы отметить, что синхронизация может происходить не только по вертикали кэша, т.е. от уровня к уровню, но и по горизонтали:

Допустим, около 100.000 пользователей работают с неким сервисом, который предоставляет им личный кабинет с фотоальбомом. Каждый пользователь решает показывать ему фото остальным, или каким пользователям показывать, а каким нет. В нашем случае такая задача сведётся к синхронизации

кэшей для разных пользователей, что и будет в наших терминах синхронизацией по горизонтали.

### **OR mapping.**

Для того чтобы перейти от данных к объектам и обратно будем использовать подход OR mapping. Пусть в нашей системе будет некий реестр, связующее звено между уровнями кэша, архитектурной концепцией и данными. Добавим в этот реестр связи между объектами и описание политик безопасности. В результате мы получим мощный инструмент для администрирования системы, управления дизайном системы и «переходник» между запросами к объекту и запросами к данным.

Таким подходом мы обеспечиваем прозрачность приложения для разработчиков, простоту понимания для администратора системы, уменьшаем время разработки и проектирования. Дополнив данный реестр визуальным редактором и генератором кода, мы успешно сможем имплементировать MDA подход к созданию приложения

Важным результатом проделанной нами работы является обобщение всей системы без потери производительности, что, на мой взгляд, ведёт к успеху проекта в целом и нашей команды в частности.

### **Часть 3. Хранилище данных.**

Организуя хранилище данных, в нашем случае следует отходить от нормальных форм, не пренебрегать избыточностью данных и строить максимально простые запросы. В этой части мы сделаем акцент на общих рекомендациях и будем максимально кратки. Итак:

- Если нам надо использовать сложные структуры, агрегированные данные – используем представления и определяем для них класс в нашем реестре.
- Используем запросы к объектам, вместо запроса к данным (SQL92) – они короче, понятнее и занимают меньше места в кэше ключей.
- Используем промежуточные таблицы для агрегации статистики и определяем для них класс в реестре.



Обновления данных в таких таблицах делаем асинхронными сервисами.

- Убираем не актуальные данные в архивное хранилище. Чем меньше актуальная база данных, тем система быстрее будет работать.
- Если мы доверяем нашему приложению и файлу реестра, а главное гарантируем целостность данных, то можно отказаться от работы с внешними ключами, они понижают производительность базы.

Используя наш подход и концепцию много времени тратить на организацию данных, не имеет смысла. Основная функция базы – хранилище данных. А всю логику мы уже имплементировали. Однако крайне важно следить за корректностью запросов, при ошибочной и слишком громоздкой архитектуре система может очень сильно понизить производительность хранилища. Перечислим узкие места, на которые стоит обратить внимание:

- Связи между объектами.
- Рекурсия.
- Вычисляемые свойства.
- Низкие уровни кэширования.
- Открытые запросы к объектам (запросы без условия и политик).

## **Вывод**

Итак, мы ввели такие термины как: Контент, Рубрикатор, Группа, связи между этими понятиями, типы Контента, статистика Контента, политики безопасности, Администратор приложения.

В дополнение к вышеперечисленному мы знаем структуру кэша и правила организации данных.

Несомненно, концепция, изложенная в данном докладе, может явиться некоторым единым документом, определением для нашей системы и, опираясь на это определение (конечно, расширенное и дополненное), мы можем приступить к разработке.

## **Abstract**

Rapidly developing software outsourcing market requires more and more high level professionals as the time goes. Today many HR professionals say that the lack of IT specialists in all fields and companies is growing. The most experts also add that such a state of things holds back of the industry as a whole. Many developers complain about lowering quality of young specialists and also about discrepancy between education programs and real market requirements. Here you will find the overall analysis and our suggestions how the named problems can be solved.

## **Тезисы**

Бурно развивающийся рынок разработки заказного программного обеспечения с каждым годом требует все большее число высококлассных работников. В современных условиях многие специалисты по рекрутингу отмечают возрастающую нехватку ИТ-сотрудников в различных компаниях. Большинство аналитиков отмечает, что подобная ситуация является одним из сдерживающих факторов для развития области в целом. Также отмечается, что существенно снизилось качество подготовки молодых специалистов, а существующие учебные программы не соответствуют требованиям рынка. В нашей работе проделана попытка проанализировать причины сложившейся ситуации, а также рассмотреть пути их решения.

**Keywords:** IT, HR, questions, answers, problems

## **1. Введение**

Проблемы с подготовкой квалифицированных специалистов широко известны и постоянно обсуждаются на различных уровнях государственной власти, предприятий, учебных заведений. Отмечается, что уровень образования с каждым годом падает [4], страна получает все меньшее число

квалифицированных кадров, что является в свою очередь сдерживающим фактором для роста экономики.

Наибольшую заметность такие проблемы приобретают в области информационных технологий. ИТ на текущий момент одна из бурно развивающихся областей российской промышленности, здесь осваиваются новые инструменты, области и технологии. В результате система образования не поспевает за требованиями рынка.

Проблема усугубляется снижающимся числом выпускаемых специалистов. По оценкам некоторых экспертов, потребность ИТ-компаний Санкт-Петербурга составляет примерно шесть тысяч человек в год, при этом число выпускаемых специалистов составляет число порядка двух тысяч в год. В результате ИТ-компании ощущают кадровый «голод» [1], замедляющий дальнейшее развитие области в целом.

## **2. Что требует рынок**

До недавнего времени компаниям ИТ-сектора требовались готовые специалисты, с опытом работы, которых можно быстро подключить к решению производственных задач. Отсюда проистекает требование знание современных платформ, языков программирования, а так же инструментальных средств.

При этом разнообразии специальностей востребованных рынком достаточно широко это бизнес и системные аналитики, инженеры по тестированию, системные интеграторы, администраторы, конфигураторы, управленцы, маркетологи и т. д. Однако во «внешнем» мире существует четкое заблуждение, что в ИТ работают только программисты. Список специальностей довольно широк, при этом еще существует разделение по технологиям, в результате спектр еще больше расширяется.

По некоторым оценкам непосредственно программисты составляют только половину от общего числа востребованных специалистов. Должностные обязанности программистов довольно широки и кроме непосредственно кодирования включают в себя разработку архитектуры, платформенного

решения, выделение программных модулей и интеграционных приложений, поддержку их работы и даже программное сопровождение проекта. Кроме этого представители данной профессии ответственны за составление технической документации по разработанному программному обеспечению [5].

Наряду с опытом работы немаловажным фактором является наличие высшего образования. Несмотря на активное вовлечение студентов в процесс разработки, эксперты отмечают, что примерно 70 % сотрудников ИТ-компаний имеют высшее образование, причем большую часть из них составляют специалисты и магистры. Доля бакалавров и выпускников техникумов, которые, по мнению авторов и должны составлять основную часть коллектива, относительно мала и по некоторым оценкам составляет от 10 до 15 процентов.

Большинство крупных ИТ-компаний уже давно отошло от «наколенной» сборки продуктов к процессному производству. При этом компании, как правило, проходят сертификацию процесса производства на соответствие признанным международным стандартам. Это способствует привлечению новых зарубежных заказчиков. Некоторые компании активно внедряют новейшие технологии производства программного обеспечения, позволяющие им занимать лидирующие позиции на рынке. Как следствие дополнительное требование к потенциальным работникам — знание современных методологий производства программного обеспечения, понимание жизненного цикла программного обеспечения, управления требованиями, изменениями, планирования, управления рисками и так далее.

Для компаний занимающихся производством программного обеспечения на заказ, важным фактором является свободное владение иностранным языком, преимущественно английским [3].

Также не маловажным моментом являются зарплатные ожидания кандидатов, особенно это относится к специалистам без опыта работы. Авторам не раз доводилось беседовать со

студентами высших учебных заведений, а также с выпускниками компьютерных специальностей техникумов. Уровень таких специалистов, как правило, не высок. Знания в основном сосредоточены в умении кодировать. При этом зарплатные ожидания на период испытательного срока у них находятся на уровне равном или чуть большем, чем у среднего специалиста с опытом работы не менее года. Интересным является тот факт, что подобные специалисты, в большинстве случаев находят удовлетворяющие их позиции.

Современный ИТ-продукт практически невозможно производить в одиночку, безусловно, на рынке встречаются подобные исключения, однако, в большинстве случаев это результат труда команды людей. Команда представляет коллектив профессионалов, взаимодействующих друг с другом, для достижения общей поставленной цели, при этом члены команды, как правило, обладают уникальными навыками. Как результат появляются новые требования к кандидатам — умение работать в коллективе, открытость, дружелюбие, способность к общению.

Подводя краткий итог, можно отметить, что приведенный выше список, нельзя считать всеобъемлющим. В нем авторы попытались показать наиболее востребованные современным ИТ-рынком навыки и умения.

### **3. Что предлагают вузы**

Давайте попытаемся проследить, что в сложившихся условиях могут предложить высшие учебные заведения.

Россия традиционно считается страной с сильной фундаментальной подготовкой. Это отмечают все, включая западных партнеров. В процессе обучения выпускник получает хорошую базу в естественных науках. Однако впоследствии выясняется, что пять лет специалиста учили не тому, было нужно. Если посмотреть учебные программы современных специалистов, то можно отметить, что фокус делается на фундаментальные области, рассматриваются базовые вещи. Такой фундамент, является необходимым условием высшего

образования. Позволяет всесторонне развить подготавливаемого специалиста, а также научить его мыслить и решать задачи из широкого круга.

Положительным, по мнению авторов, является введение в России двух ступенчатой системы образования, которое проходит в рамках «Болонского процесса». Предполагается, что уже после четырех лет образования новоиспеченные бакалавры приступят к профессиональной деятельности, при этом на некоторое время прекратят процесс получения высшего образования, что позволит пополнить ряды молодых специалистов, востребованных рынком. В силу ряда причин в большинстве случаев, студенты после получения бакалаврского диплома, продолжают обучение до степени магистра. Это в некоторой степени оттягивает их полноценное вовлечение в производственный процесс.

В последнее время в области высшего образования намечаются единичные случаи сотрудничества высшей школы и бизнеса, имеют место создание, разработка и реализация учебных курсов и в некоторых случаях, даже создание собственных учебных кафедр. Однако анализ показывает, что подобные шаги оказывают малое влияние на отрасль в целом. Подобные шаги могут помочь определенной компании решить локальные кадровые проблемы, при этом производительность таких кафедр весьма ограничена.

#### **4. Проблемы и возможные пути их решения**

Как обычно существует два четких пути развития области интенсивный и экстенсивный. К экстенсивному можно отнести стремление ИТ-компаний открывать центры разработки в регионах. Это чем-то напоминает опыт с целинными землями. Как правило, выбираются удаленные регионы с сильным техническим образованием, развитой инфраструктурой и низкими зарплатными ожиданиями. При этом обвинять бизнес в подобном пути развития было бы не правильно. Во-первых, это позволяет компаниям выживать в сложившихся условиях, во-вторых, таким образом, в регионах появляются

высококласные специалисты с опытом реальных проектов, которые, в конечном счете, будут являться ядром будущих разработок.

Однако стоит отметить, что ресурсы страны не безграничны и развитие в регионах не может продолжаться бесконечно, рано или поздно возникнут такие же проблемы, какие были в начальных центрах разработки.

По мнению авторов, для организации эффективной подготовки новых кадров, решающую роль играет наличие эффективной обратной связи направленной от ИТ-компаний, к научным заведениям. В результате такого общения учебные заведения должны четко представлять, сколько и каких специалистов потребуется промышленности в ближайшем будущем. Это необходимо для своей временной корректировки учебных планов, открытия дополнительных факультативных курсов. Это не секрет, что постановка учебного курса в вузе в среднем занимает три года. За это время разрабатывается программа, подготавливается необходимый методический материал.

Стоит отметить, что в условиях, когда на подготовку учебного курса тратится значительное время и быстро меняющимся требованиям рынка, будет существовать разрыв между образованием и требованием предприятий. Авторы не видят в таком положении дел серьезных проблем. Дело в том, образование служит отправной базой на основании, которой специалист строит свою будущую карьеру. Поэтому детальное изучение инструментов вряд ли вообще должно входить в академическую программу, их обзор нужен лишь постольку, поскольку они помогают осваивать дисциплины общего характера [6].

В больших объемах должно происходить сотрудничество компаний с учебными заведениями. Причем не только по ИТ-специальностям. Речь идет от диверсификации требований к направлениям специализации будущих сотрудников. Эта рекомендация базируется на нескольких фактах. Выпускники «компьютерных» кафедр, как правило, обладают таким же базовым образованием, как и выпускники кафедр, где

компьютерные технологии не являются профилирующими. При этом будущие ИТ-специалисты обладают повышенными требованиями к уровню компенсаций за свой труд, в силу пристального влияния рынка. Их коллеги с непрофильных кафедр, в свою очередь, обладают меньшими зарплатными амбициями. Время ввода в процесс производства новичков с ИТ-образованием находится примерно на одном уровне со временем необходимым для непрофильных специалистов без опыта работы.

К сдерживающим факторам в такой диверсификации можно отнести обыкновенную боязнь непрофильных специалистов, не желание даже попробовать свои силы и пройти собеседование по «незнакомой» специальности. Авторам не раз приходилось наблюдать и общаться со студентами различных не «компьютерных» специальностей, при этом в качестве основного фактора, сдерживающего специалиста от подачи резюме на ИТ-должность, озвучивались такие аргументы как — незнание предметной области.

К парадоксальным фактам, стоит отнести то, что в сложившихся условиях учебные заведения не сильно заинтересованы в подготовке специалистов, востребованных рынком. Связанно это с несколькими причинами. Во-первых, заказчиком образовательных услуг является учащийся, а вовсе не его будущий работодатель. Во-вторых, модификация учебных программ и методических материалов довольно трудоемкая и медленно решаемая задача. В-третьих, образование в том виде, в котором оно существует на текущий момент, вполне востребовано и нет существенных предпосылок, по которым требовалось бы что-то менять. По большому счету высшая школа сегодня работает по экономической модели, сохранившейся с советских времен, при которой вся ответственность за обучение технических кадров возлагалась на государство — кого, сколько и как нужно подготовить. Но раньше государство являлось и потребителем этих кадров, и поэтому работала обратная связь, необходимая для постоянного контроля качества обучения. А сегодня эта связь фактически разорвана [7].



Столкнувшись с необходимостью подготовки кадров софтверные компании, приступили к самостоятельному решению проблем. Можно выделить несколько направлений, по которым ведется работа. Это организация внутренних учебных курсов. Выглядит это следующим образом, в рамках компании сотрудники подготавливают некоторый учебный материал, который впоследствии в сжатые сроки читается новичкам, в рамках подготовки их к будущей профессиональной деятельности в рамках компании. Такой подход хорошо зарекомендовал себя как временное средство в условиях острой нехватки необходимых специалистов. В рамках такой модели, специалисты с реальным опытом, передают новейшие знания и наработки в области ИТ-производства. Однако такой подход решает только локальную проблему подготовки кадров, да и то временно. По мнению авторов, подобный подход, необходим, но только в рамках производственной практики, когда потенциальные специалисты могут ликвидировать разрыв между полученным академическим образованием и существующими рыночными реалиями.

Также не стоит забывать, что для организации эффективной подготовки специалистов, сотрудники учебных заведений должны иметь доступ к опыту использования современных технологий в процессе производства программного обеспечения. В условиях, когда академические вузы не ведут научные исследования в областях производства ИТ-продуктов, не возможна эффективная организация процесса обучения. Более того, учебные заведения также испытывают кадровый голод, по различным оценкам, средний возраст профессорского преподавательского состава колеблется от 37 до 45 лет. Причем в качестве основных проблем выделяют низкий уровень оплаты труда, преподавателей. Безусловно, такая проблема имеет место, нередко встречаются ситуации, когда студенты дневного обучения, имеют заработок выше, чем их преподаватели, при этом мало в чем уступают в знании современных технологий своим именитым учителям. По мнению авторов, проблема оплаты труда преподавателей, безусловно,

важна и актуальна, однако она второстепенна. Решать подобные проблемы восполнения разрыва между академическими заведениями и предприятиями промышленности, необходимо совместными усилиями. Для начала деятельности в этом направлении, по мнению авторов необходимо, четко сформулировать требования к современным ИТ-специалистам. На текущий момент это под силу только представителям бизнеса. Сформировать количественную оценку числа требуемых специалистов, а также разделение их по областям знаний и специальностям. Это позволит учебным заведениям сфокусировать усилия на проблемных областях. При этом на последних курсах обучения необходимо проводить активную специализацию будущих выпускников на программистов, архитекторов, аналитиков, управленцев и т.д. При этом должна присутствовать командная работа, когда будущие специалисты участвуют в разработке живых проектов. В идеале должен сформироваться готовый коллектив разработчиков.

В требованиях к кандидатам часто можно видеть требование наличия у соискателя ИТ-должности высшего образования. Зачастую это объясняется тем, что наличие высшего образования подтверждает, что у потенциального сотрудника имеется навык к обучению. Стоит отметить, что умение учиться, безусловно, важно в современных условиях, однако, это совсем обозначает умение соискателя применять полученные навыки на практике. В результате ИТ-компании сами срезают часть потенциальных кандидатов на работу. Мировой опыт показывает, что в большинстве зарубежных компаний соотношение сотрудников с высшим образованием к числу прочих составляет примерно двадцать процентов к восьмидесяти. Более того, подготовка специалиста с высшим образованием это долгий и трудоемкий процесс, а кадры нужны уже сейчас. Процесс подготовки программистов не должен занимать пять лет. Опыт авторов показывает и это подтверждается наблюдениями других специалистов, что в большинстве вузов, готовящих программистов, студенты, дойдя до определенного уровня, обычно до третьего курса,

перестают посещать лекции, предпочитая работать по специальности. Как следствие — неизбежный пропуск спецкурсов. В качестве одной из причин пропусков можно выделить чрезмерную растянутость сроков обучения [2]. При этом полученных за три года обучения знаний, вполне хватает для ведения профессиональной деятельности. Это вовсе не означает, что необходимо сократить сроки высшего образования, как это делается в некоторых странах. Это обернется «гуманитарной катастрофой». В сложившейся ситуации, по мнению авторов, стоит обратить внимание в сторону средних технических заведений, которые потенциально могут стать кузницей кадров для ИТ-промышленности.

Отмеченные недостатки в уровне владения иностранными языками возможно решить, действуя по нескольким направлениям. Необходимо всячески стимулировать изучение иностранных языков, также необходимо активно развивать навыки разговорного языка. Самое интересное, что заинтересованность есть и учебных заведений и в ИТ-компаниях.

## **5. Выводы**

Необходимо вовлекать учебные заведения в процесс разработки программного обеспечения, сокращать сроки обучения специалистов и снижать входные требования к ним. При этом большое внимание необходимо уделять организации практики в рамках учебного процесса, со специализацией на последних этапах обучения.

## **6. Ссылки**

- [1] «Кадровая ИТ-блокада Петербурга продолжается» // [http://www.cnews.ru/reviews/index.shtml?2008/03/17/292453\\_1](http://www.cnews.ru/reviews/index.shtml?2008/03/17/292453_1)
- [2] Науменко С. Как вырастить армию хороших программистов и поднять объем экспорта ПО? Взгляд «от сохи» // Компьютерра. 2005. № 4.

- [3] Батурко О. Фундаментальная стабильность // CNews. 2008, № 3. С. 102-103.
- [4] Андропова О. Второй по значимости ресурс после нефти // [http://www.ci.ru/inform14\\_04/p\\_04.htm](http://www.ci.ru/inform14_04/p_04.htm).
- [5] Сколько стоит Программист Java в 9 городах России // Рынок труда. Мнение экспертов. [http://www.e-prof.ru/jurnal/rynok\\_truda\\_mneniya\\_ekspertov/programmer\\_java.htm](http://www.e-prof.ru/jurnal/rynok_truda_mneniya_ekspertov/programmer_java.htm), 29.07.08.
- [6] Колесов А. Подготовка ИТ-кадров: что мы хотим от аутсорсинга // <http://www.visual.2000.ru/kolesov/pcweek/2005/50321edu.htm>.
- [7] Колесов А. Академическая подготовка ИТ-специалистов. Есть проблемы? // PC week. 2004, № 32. С. 41.

# **Опыт построения и использования практики количественного проектного управления, соответствующей требованиям модели СММІ v1.2 и согласованной с бизнес-целями организации**

**Станислав Калканов**  
**Люкссофт**  
**Москва, Россия**  
**skalkanov@luxoft.com**

**Николай Быков**  
**Люкссофт**  
**Москва, Россия**  
**nbykov@luxoft.com**

## **Тезисы**

Когда организация начинает реально использовать количественное управление проектами, соответствующее high maturity требованиям модели СММІ или когда организации необходимо количественно оценить эффективность процесса разработки ПО и отдачу от инвестиций в его развитие, одним из ключевых факторов успеха является использования подходящих средств автоматизации измерений и системы Статистического Анализа и Управления (САУ). Эта система должна адекватно учитывать особенности производственного процесса организации, а также позволять получать данные, необходимые для расчета количественных бизнес целей организации.

Система САУ компании Люкссофт реализована как приложение Microsoft Access. Система интегрирована с применяемыми в проектах измерительными формами, основанными на Microsoft Excel и специализированных измерительных библиотеках. Система используется для накопления проектных измерений, анализа проектных метрик и причин их вариаций, анализа стабильности процессов разработки, подготовки метрической отчетности на уровне проектов, программ и производственных центров, а также для поддержки статистических бейзлайнов.

Данная презентация знакомит с основами организации системы САУ Люкссофт и рассказывает, как данная система используется для эффективного сбора, анализа проектной статистики и подготовки отчетности по количественным индикаторам процесса разработки ПО.

Презентация отвечает на вопросы как:

1) обеспечить сбор и обработку статистических данных для анализа статистических гипотез и подготовки статистически значимых наборов данных;

2) вычислять количественные индикаторы и готовить статистические бейзлайны уровня организации;

3) готовить статистические данные для количественного управления в проектах.

**Keywords:** CMMI, количественное управление проектами, бизнес-цели, метрики.

# **Practical experience of quantitative project management practice development and usage to meet CMMI v1.2 high maturity requirements and organizational business objectives**

**Stanislav Kalkanov**  
**Luxoft**  
**Moscow, Russia**  
**skalkanov@luxoft.com**

**Nikolay Bykov**  
**Luxoft**  
**Moscow, Russia**  
**nbykov@luxoft.com**

## **Abstract**

When organization starts its way to high-maturity CMMI levels or just has strong intention to quantitatively understand software development process effectiveness and ROI of process improvement it's necessary to use appropriate Statistical Process Control tool aligned both with organizational culture and processes as well as with organizational business objectives.

Luxoft's SPC tool is a MS-Access based application for support and automation of all SPC-specific activities of. It's integrated with Microsoft Excel-based measurement forms used in projects to collect data, analyze causes of variation, test stability, report on all levels and maintain statistical baselines.

The presentation will discuss the creation of a Luxoft proprietary SPC tool and how it is used to collect, analyze and reporting quantitatively managed indicators the easy way.

It gives an understanding of how:

- 1) provide statistical data analysis to prove statistical hypotheses and define statistically proven data sets
- 2) calculate SPC indicators and generate performance baselines for the organization
- 3) prepare statistical data for quantitative project management.

**Keywords:** CMMI, High-Maturity, Measurements and Analysis, Statistical Process Control, Business Goals.

## **1. Introduction**

When an organization starts to implement CMMI high maturity practices, especially OPP and QPM, it's very important to create a

functional but still easy to use statistical process control support system. It should be:

- flexible to support metrics data collection in heterogeneous environment;
- provide all statistical functionality necessary;
- offer good reporting possibilities to provide all necessary statistical views on the project, program and organizational levels;
- allow to calculate software engineering metrics in a form aligned with organizational business objectives.

We will address how SPC support system can be established based on example of Luxoft SPC support tool used for quantitative management support in Luxoft more than 7 years.

The presentation will elaborate on the tooling side:

- How overall measurement & analysis practice including SPC approach can be establish across the organization
- how to establish flexible approach of collecting measurement results from different projects in one single database
- how to select statistically-relevant data and what tests can be done for that
- how to calculate SPC indicators and how to form useable Process Capability Baselines based on indicators for different delivery centers as well as for the whole organization
- what reports are created for project, program and organizational levels
- how to make statistical reports for the business and marketing purposes

And on appraisal/CMMI considerations:

- how the tool implements selected CMMI high maturity practices of OPP/QPM



- how the tool eases the implementation of CMMIs subpractices to fulfill SEIs raised bar.

## **2. Measurement & analysis process**

Some details on the way Luxoft does measurements and how it impacts the tool:

Project measurements are made with the help of Luxoft proprietary tools and libraries as well as standard features of defect tracking systems like ClearQuest or Jira. All the project measurement are collected and stored in pre-defines xls files. After project is finished, all the final data come into Luxoft proprietary SPC tool for statistical analysis. As a result final statistical report for the project is generated and BeyondPCB (PCB stands for Process Capability Baseline) statistics set is updated. BeyondPCB set is changed during the year after every project closure but once in a year BeyondPCB is baselined. It becomes approved PCB set for the next fiscal year (both on delivery centers and company-wide levels).

## **3. SPC tool**

Luxoft SPC tool is MS Access based proprietary system for collecting, analyzing and reporting quantitative project indicators. From mathematical toolbox point of view it's based on six sigma principles. Now the analysis is based on 48 metrics and permits to have 31 indicators under statistical control.

Detailed SPC tool features:

- Import and storage measurement results for completed projects/project releases
- Storage of 17 additional "project context" attributes like delivery center name, project manager(s), project type (DEV/MNT), technologies (Java/.NET/etc.), business domain, etc.
- SPC calculations and analysis
  - o SPC indicator definition
  - o Measurement statistics for the SPC indicator definition
  - o Statistics data quality checks

- Visual control of statistics data & SPC indicator quality
  - Sigma-deviation diagram control
  - Sigma-based tests control
    - Overall status control
    - Primary statistics in anomaly control
  - Moving Rang control
    - Overall status control
    - Primary statistics in anomaly control
- Control of statistical data representativeness
- Statistics/data stratification and sub grouping
- Statistical data removal based on SPC-criteria
- Statistical data removal from non-basis projects set
- Statistical data removal due to informal criteria
- Boundary values removal
- SPC indicator calculation (approx. 15 methods)
- SPC indicator normal distribution calculation
- SPC indicator critical data ranges calculation
- Create report in xls-format for further visual XmR-chart analysis
- Process Capability Baseline (PCB) calculations
  - SPC indicator value updates in PCB after SPC calculations
  - SPC indicator value updates in beyond-PCB after SPC calculations
- Typical reports generation
  - Strata indicators and their values
  - Quality of indicators in strata
  - Projects in basic set of projects
  - Projects in particular strata
  - Project parameters (size in SLOCS & FP, duration, project team size)
- Project quantitative analysis reports
  - Set of quantitatively tracked metrics for the project and their values

- Aggregative “business” metrics values (economy, project effectiveness, product quality, process quality)
- Cumulative quality of the project
- Xls documents generation in special template to be used for as appendices to the project closure report with statistical analysis results
- Project benchmarking
  - Comparison between project indicators with corresponding PCB values for the delivery center
  - Comparison between project indicators with corresponding beyondPCB values for the delivery center
  - The same in comparison with company-wide PCB/beyond PCB
  - Comparison with aggregative “business” metrics values (economy, project effectiveness, product quality, process quality)
  - Comparison with the best historical values
- Project estimation precision analysis

#### **4. Conclusions**

Full featured and aligned with organizational needs Statistical Process Control tool is essential pre-requisite to build CMMI high maturity compliant quantitative project management practice. It helps to save a lot of money on data collection, processing and analysis and provide data necessary for quantitative monitoring of organizational business objectives.

With current SPC tool it's possible to support all SPC analysis needs of 3000+ distributed company. It's used more than 7 years and proves it's efficiency.

# **CMMI Appraisal – Cost Control and Extra Business Benefit**

**Stanislav Kalkanov**

**Luxoft**

**Moscow, Russia**

**email: SKalkanov@luxoft.com**

**Grigory Gusev**

**Luxoft**

**Moscow, Russia**

**email: GGusev@luxoft.com**

## **Abstract**

Companies invest much time and resources in CMMI Appraisals and often get only formal “certificate” for marketing and sales purposes. Sometimes that “certificate” turns out to be too expensive to be repaid by business benefits. At the same time, a company may consciously control the cost of the CMMI Appraisal as a project and get mutual benefit both from the Appraisal process and business results. Being thoughtfully organized and planned, the Appraisal procedure may result in valuable process improvement opportunities provided by the Appraiser’s external view, and higher process awareness and culture within the company.

Luxoft has gained a wealth of 7-year experience in SEI CMM, CMMI v1.1, and CMMI-Dev v1.2 successful Appraisals. The article summarizes lessons learnt by Luxoft on its way to CMMI excellence and warns of the typical problems with the first and the following CMMI Appraisals.

**Keywords:** CMMI v1.2, CMMI Appraisal, Appraisal Process, Business Benefit.

# **Опыт оцениваний по модели СММИ – как сократить затраты на оценивание и получить дополнительные выгоды для бизнеса**

**Станислав Калканов**  
**Люксофт**  
**Москва, Россия**  
**email: SKalkanov@luxoft.com**

**Григорий Гусев**  
**Люксофт**  
**Москва, Россия**  
**email: GGusev@luxoft.com**

## **Тезисы**

Организации часто несут неоправданно большие издержки на подготовку и проведение оцениваний (Appraisal) по модели СММИ и при этом получают только формальный «сертификат», которым могут похвастаться отделы маркетинга и продаж. Иногда такой «сертификат» оказывается слишком дорогим и просто не окупается. Не многие задумываются о том, что можно контролировать стоимость СММИ оцениваний как проектов, и о том, что само оценивание может стать полезным для бизнеса компании. При тщательном и продуманном планировании процедура оценивания может открыть новые возможности по улучшению процессов (невидимые изнутри, но видимые для внешнего оценщика) и повысить процессную культуру и процессную осведомлённость в компании.

В течение 7ми лет успешных оцениваний по моделям СММ, СММИ версии 1.1 и СММИ-Dev версии 1.2 Люксофт накопил богатейший опыт организации этого процесса. Данная статья обобщает люксофтовские уроки: типичные проблемы первого и последующих СММИ оцениваний и пути решения этих проблем.

**Keywords:** СММИ

## **1. Введение**

Проект по подготовке и проведению оценивания по модели СММИ занимает от года до нескольких лет и требует значительных денежных затрат. Особенно сложно приходится тем компаниям, которые выполняют такой проект первый раз. Незнание специфики оценивания и отсутствие опыта выполнения подобных проектов приходится компенсировать существенным увеличением затрат средств и увеличением сроков подготовки к оцениванию.

Компания Luxoft имеет большой опыт проведения формальных и неформальных аудитов по ISO, CMMI, RUP, Agile-методологиям. По моделям CMM/CMMI оценивания проходят с 2002 года, и в ноябре 2007 года компания стала первой в Европе, получившей 5-ый, высший уровень зрелости по последней версии – CMMI-DEV v1.2.

Опыт, накопленный с 2002 года, и особенно, полученный в ходе последнего оценивания 2007 года, является уникальным для России и стран СНГ и поможет избежать типичных ошибок и получить дополнительные выгоды от использования модели CMMI.

При проведении официального оценивания (appraisal) по модели CMMI, помимо получения формального «сертификата», обычно ставятся цели минимизации затрат и получения от этой процедуры практической пользы, как и от любого внешнего аудита: предложений по улучшению процессов, знаний о передовом опыте других организаций. В рамках данного доклада будет рассмотрено, как эти задачи решались в Luxoft.

## **2. CMMI в Люксофт**

Краткая история модели CMMI в Luxoft:

- 2002 – оценивание по CMM 4го уровня
- 2003 – Первое в Европе одновременное оценивание по CMM и CMMI версии 1.1 5го уровня
- 2007 – Первое в Европе оценивание по CMMI версии 1.2 5го уровня
- 2008 – Первая и единственная компания – разработчик ПО в странах СНГ - официальный партнёр Software Engineering Institute(SEI), организации-разработчика модели CMMI.

Проект CMMI 2007 по подготовке и проведению оценивания проводился как внутренний проект, спонсировался на уровне СЕО компании, и выполнялся силами Центра Качества и сертифицируемого подразделения (Центра Специальных Разработок Aerospace).

Центр Качества осуществлял общую координацию проекта, искал и выбирал официального оценщика (Appraiser) сертифицированного SEI, организовал общие тренинги по

модели СММІ, выполнял работы по company-wide практикам и проводил внутренние аудиты. Группа развития процессов (SEPG) ЦСР Aerospace выполняла работы по практикам производственного центра, отвечала за подготовку проектов и сотрудников производственного центра и проводила специализированные тренинги.

Из состава Центра Качества и ЦСР Aerospace была сформирована внутренняя команда оценки (Appraisal Team) из 7-ми человек, которую возглавлял внешний официальный оценщик.

Предварительная подготовка и тренинги начались в декабре 2006 года, активная фаза подготовки проектов – в марте 2007 года, а официальные аудиты с участием внешнего оценщика (SCAMPI C – SCAMPI A) проходили с 16 августа по 16 ноября 2007 года. Общие трудозатраты по проекту составили порядка 8 000 человеко-часов. Появление серьёзных проблем успешно предотвращалось в рамках процесса управления рисками с учетом предыдущего опыта прохождения оцениваний и выполнения подобных проектов для сторонних организаций, подробнее об этом в следующем разделе.

### **3. Особенности и ограничения проекта**

Как и любой проект, проект по СММІ обладал рядом условий и ограничений:

- Сроки – оценивание нужно было пройти до конца 2007 года
- Распределенность проекта – производственный центр располагается на трех площадках (Москва, Дубна, Омск)
- Бюджет – бюджет должен был быть минимизирован, насколько возможно
- Невозможность воспользоваться услугами оценщика, с которым компания работала в 2002-2003 годах в связи с выходом новой версии модели 1.2 и новых правил оценивания
- Скепсис сотрудников на почве модных тенденций применения гибких методологий, противопоставляющих себя СММІ

- Наличие рабочих инструкций на русском языке и некоторых ключевых специалистов, не говорящих на английском языке.

#### **4. Решение проблем и ограничений проекта**

Особенности и ограничения проекта решались следующим способом:

- Использование автоматизированной системы управления подготовкой к оцениванию - CMMI Tool, который позволяет следующее:
  - Структурированное хранение требований модели CMMI (Process Areas, Goals, Practices, TWP's)
  - Меппинг корпоративных политик и шаблонов (QMS)
  - Меппинг проектных артефактов (PIID)
  - Запись комментариев, strength, weaknesses
  - Хранение и анализ результатов оценивания

Применение автоматизированной системы позволило:

- существенно сократить время на подготовку т.н. РП матриц с формальным подтверждением соответствия требованиям модели CMMI,
- выполнять работы в распределенном режиме между Москвой, Дубной и Омском
- сократить время на ревью результатов со стороны Lead Appraiser и отработку его замечаний
- более полно и оперативно контролировать прогресс проекта
- Организация 24x7 on-line доступа Lead Appraiser как к самому CMMI tool, так и к зеркалу проектных



репозитариев. Это позволило провести Readiness Review без on-site Lead Appraiser и существенно увеличивало уровень доверия Lead Appraiser к оцениваемой компании – все проекты и все работы по подготовке были прозрачны для оценщика. Это решение также способствовало эффективной организации работ распределенной команды со стороны Люксофт.

- Организация распределенной среды для проведения видеоконференций. Использовалась система видео- и аудиоконференций Люксофт, а также Webex для проведения демонстрации на удаленных рабочих местах. Это позволило выполнять все интервью из одной основной локации – Москвы и сократить время и средства на перелеты.
- Популяризация СММІ модели и мотивация сотрудников. Для популяризации активностей по СММІ и мотивации сотрудников компании использовались следующие техники:
  - Lead Appraiser выбирался не по принципу – кто меньше спрашивает, а по принципу – кто может дать БОЛЬШОЙ бизнес value, поделившись своим опытом и подсказав оптимальный вариант реализации требований модели СММІ с учетом бизнес и технологических особенностей конкретного производственного центра. Общение с Lead Appraiser по мнению большинства участников было интересным и практически полезным, что существенно положительно отразилось на мотивации вовлеченных в проект сотрудников.
  - Проведение brainstorm-ов на тему: “цели компании, персональные цели, практики компании, модель СММІ – как практики компании и применение модели СММІ позволяют достигать целей компании и персональных целей сотрудников”. На этих семинарах сотрудники вместе со специалистами по процессам и модели СММІ обсуждали стратегические цели

компании, формулировали свои персональные цели и связывали достижение целей с практиками, применяющимися в компании, в частности с СММИ практиками.

- Проведение дополнительных углубленных тренингов по следующим направлениям:
  - СММИ модель
  - Практическая реализация СММИ в компании, меппинг повседневных процессов и активностей на модель
  - Количественное управление и статистика (для менеджеров, тест менеджеров, тим-лидов и всех интересующихся)
  - Принципы проведения оценивания, SCAMPI (для принимающих участие в интервью)
  - СММИ Tool (для членов внутренней команды по оцениванию)
- Перевод русскоязычных инструкций и синхронный перевод во время проведения интервью. Некоторые ключевые рабочие инструкции были полностью либо частично переведены на английский язык. На эти работы были заранее запланированы ресурсы технических писателей и ревьюеров. Также в команду внутренних оценщиков были включены сотрудники, уровень знания английского которых позволял выполнять синхронный перевод интервью с теми ключевыми сотрудниками компании, которые не обладали разговорным английским.
- Внимательный выбор официального оценщика:
  - Большое количество рассмотренных кандидатов (около 30 человек), рассылка RFP и анализ proposals (около 20), личные

встречи с кандидатами, вышедшими в финал (3 человека).

- Выбор оценщика с учётом не только цены, но и «качества» - то есть его репутации и опыта, а также мнения сотрудников SEI.
- Подписание контракта с оценщиком на поэтапной основе

## **5. Уроки и выводы**

- Инвестиции в систему удаленных веб-конференций и оснащение переговорных комнат окупают себя и позволяют сэкономить существенные суммы на уменьшении количества командировок.
- Возможное сокращение сроков on-site визитов Lead Appraiser. Практика показала, что за счет автоматизации и политики открытости продолжительность финального оценивания может быть сокращена до 10-12 рабочих дней.
- Инвестиции в автоматизированные системы поддержки производственных процессов окупают себя – проще получать необходимые подтверждения использования практик, снижается “бумажный” документооборот, повышается эффективность и не возникает демотивация персонала по поводу необходимости заполнения “документов для CMMI”.
- Инвестиции в дополнительное обучение и мотивацию сотрудников, в т.ч. связывание персональных целей сотрудников с целями компании, окупаются и приносят ощутимый положительный мотивационный эффект
- При планировании работ нужно учитывать внутренние процедуры SEI, оказывающие влияние на сроки проекта.
- Опыт привлечения не дешёвого, но очень квалифицированного оценщика признан успешным: получено множество предложений по дальнейшему совершенствованию процессов, в том числе за пределами компетенции CMMI, руководство и рядовые сотрудники компании получили много полезной информации об опыте

других компаний, существенно повысился уровень связанности СММІ практик с бизнес-целями компании.

## **6. Заключение**

Главным результатом можно считать то, что основные цели проекта были успешно достигнуты (оценивание проведено в установленные сроки, соблюден бюджет проекта), а коммерческие проекты, которые подвергались официальному оцениванию, не пострадали: не были нарушены сроки и внешние бюджеты. Также в рамках проекта была существенно развита культура применения практик СММІ и популяризирована сама модель.

С учетом уроков проекта и накопленного опыта стоимость последующих «сертификаций» может быть снижена.

В связи с тем, что теперь СММІ оценивания должны проходить каждые 3 года, компаниям, которые хотят поддерживать свои «сертификаты» имеет смысл формализовать и постоянно совершенствовать саму процедуру оценивания.

# **Efficiency Evaluation of Cryptographic Protection of Information in Enterprise Applications**

**Sergey Avdoshin**  
**Head of Software Engineering**  
**Department,**  
**State University – Higher**  
**School of Economics, Russia**  
**email: savdoshin@hse.ru**

**Alexandra Savelieva**  
**Lecturer,**  
**State University – Higher**  
**School of Economics, Russia**  
**email: asavelieva@hse.ru**

## **Abstract**

We introduce a complex approach to evaluating cryptographic protection efficiency. Classically, the research has mostly focused on information system security as a whole, whereas cryptographic tools evaluation techniques have not received as much attention. The main thread of our work is the development of mathematical models of threats to analyze the security of cryptographic systems based on various types of attacks that the cryptographic system is exposed to. The approach allows to build an economic rationale for investments to cryptographic systems and to provide sound arguments for implementing an information security strategy.

**Keywords:** cryptographic system, cryptanalysis, threat modeling.

# **Оценка эффективности криптографической защиты информационных ресурсов в корпоративных системах**

**Сергей Авдошин**  
Государственный университет –  
Высшая школа экономики  
Руководитель отделения  
программной инженерии  
email: savdoshin@hse.ru

**Александра Савельева**  
Государственный университет –  
Высшая школа экономики  
Преподаватель  
email: asavelieva@hse.ru

## **Краткая аннотация**

В данной статье предлагается комплексный подход к оценке эффективности защиты информационных ресурсов предприятия, обеспечиваемой криптографическими средствами. Разработанная методика позволяет провести экономическое обоснование расходов организации на обеспечение информационной безопасности и сделать обоснованный выбор мер и средств криптографической защиты. В основе методики лежит формализованный процесс анализа надежности криптосистемы в определенном контексте использования и математическая модель угроз безопасности защищаемых информационных ресурсов.

**Ключевые слова:** СКЗИ, криптоанализ, моделирование угроз.

## **1. Введение**

**Средства криптографической защиты информации (СКЗИ)** представляют собой средства вычислительной техники, осуществляющие криптографическое преобразование информации для обеспечения ее конфиденциальности и контроля целостности [1].

При оценке эффективности СКЗИ важнейшим критерием считается криптостойкость. Такой подход не учитывает других важных требований к криптосистемам. Обоснованный выбор методов криптографической защиты информации для конкретных информационных систем должен опираться не только на криптостойкость, но и на другие критерии эффективности, такие как (см. [2]):

- минимальный объем используемой ключевой информации;
- минимальная сложность реализации (в количестве машинных операций);
- стоимость;
- высокое быстродействие.

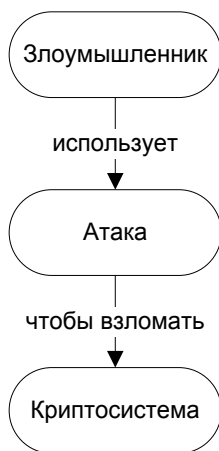
Анализ публикаций в открытом доступе показал, что подходящие методики оценки эффективности криптографических систем до сих пор не разработаны (к аналогичным выводам пришли авторы работы [2]). Исключением является статья В.П.Иванова [3], в которой эффективность криптографических средств защиты предлагается оценивать с использованием математического аппарата теории массового обслуживания и теории катастроф на основе *вероятностно-временной группы показателей*, в числе которых:

- среднее время безопасного функционирования защищаемой системы;
- время безопасного функционирования защищаемой системы с вероятностью НСД не выше заданной;
- экономическая эффективность созданной системы защиты информации.

Выбор показателей эффективности представляет интерес, однако методика имеет ряд критических недостатков, которые делают невозможным ее применение на практике для оценки современных СКЗИ. В первую очередь это границы применимости: методика подходит только для оценки криптосистем, принадлежащих по классификации Ж.Брассара [4] к классу *криптосистем ограниченного использования*, стойкость которых основывается на сохранении в секрете алгоритмов зашифрования и расшифрования. Однако, согласно фундаментальному допущению Кирхгоффа [5], стойкость криптосистемы должна основываться не на секретности алгоритмов зашифрования и расшифрования, а на секретности некоторого значения, которое называется ее *ключом*. Все современные криптосистемы построены по этому принципу, и исследования их надежности всегда должны проводиться в предположении, что потенциальному противнику о криптосистеме известно все, за исключением используемого ключа.

Еще одним недостатком описанной в работе [3] методики является то, что она не учитывает зависимости эффективности криптосистемы от условий ее использования. Очевидно, эффективность одной и той же криптосистемы в разных контекстах может существенно отличаться. Среда функционирования системы накладывает определенные ограничения на возможные сценарии атак. Простая модель сценария атаки представлена на рис. 1 [6]. Она включает три элемента, каждый из которых оказывает влияние на общую картину угроз. Типы атак, которым подвержена криптосистема, зависят от навыков, уровня доступа, бюджета и других характеристик потенциальных злоумышленников [7]. Информация, подлежащая защите, определяет возможных злоумышленников, которые

могут осуществлять попытки взлома в целях нарушения конфиденциальности, целостности или доступности (во избежание избыточности из модели исключен элемент «Защищаемые ресурсы», который задается неявно - через элемент «Злоумышленник»).



**Рисунок 1 - Сценарий взлома  
криптосистемы**

Использование СКЗИ, обеспечивающих устойчивость к взлому ниже некоторой «фоновой» вероятности, является экономически неоправданным [8]. Например, если вероятность выхода компании из бизнеса равна  $2^{-30}$  (менее чем один из миллиона), то есть ли смысл для защиты информации, которая может нанести компании ущерб, сопоставимый с кризисом рынка, использовать алгоритм, вероятность вскрытия которого за приемлемое время составляет  $2^{-200}$ ?

Наиболее эффективным при выборе и оценке криптографической системы считается использование экспертных оценок и имитационное моделирование [2]. Разработаны методы и средства, позволяющие построить модели угроз и уязвимостей информационных систем и на основе анализа рисков получить количественную оценку соотношения потерь от угроз безопасности и затрат на создание системы защиты. В работах Н.Кукановой [9, 10] описаны методы и инструменты анализа и контроля информационных рисков: британский CRAMM [11] (компания Insight Consulting, подразделение Siemens), американский RiskWatch [12] (компания RiskWatch) и российский ГРИФ [13] (компания Digital Security).



Эти инструментальные средства полезны специалисту при проведении аудита систем обеспечения безопасности предприятия, однако они не учитывают специфики СКЗИ.

Для оценки эффективности СКЗИ необходимы методики, позволяющие принимать во внимание взаимосвязь типов криптосистемы, потенциальных злоумышленников и возможных атак на защищаемые информационные ресурсы.

## **2. Постановка задачи**

Задача состоит в разработке методики анализа эффективности криптографической системы с учетом того, каким угрозам защищаемая информация будет подвергаться со стороны злоумышленников.

В качестве исходных данных для проведения оценки эффективности криптографической защиты информационных ресурсов используются данные об особенностях реализации исследуемой криптосистемы и типах потенциальных злоумышленников. Анализ эффективности осуществляется на основе финансово-экономических показателей СКЗИ. Оценка эффективности создает предпосылку для определения соответствия криптосистемы потребностям организации, принятия решений и осуществления мер по отказу от СКЗИ, не обеспечивающих необходимый уровень защиты, и внедрению систем, достигших наилучших значений показателей.

Для решения поставленной задачи необходимо:

- Формализовать процесс оценки эффективности криптографической защиты;
- Разработать математическую модель угроз безопасности информационных ресурсов, защищенных с использованием криптографических средств.
- Провести анализ существующих методов оценки СКЗИ с экономических позиций.
- Выбрать финансово-экономические показатели, подходящие для экономической оценки инвестиций в СКЗИ.

## **3. Процесс оценки эффективности криптографической защиты**

Процесс оценки эффективности криптографической защиты можно представить в виде схемы на рис.2 [6].

Цель каждого этапа – получение ответа на вопрос:

- Этап 1: Какая криптосистема является объектом атаки?
- Этап 2: Кто будет атаковать эту криптосистему?
- Этап 3: Какие методы криптоанализа с наибольшей вероятностью будут использованы при осуществлении попыток взлома криптосистемы?
- Этап 4: Способна ли криптосистема противостоять таким атакам?
- Этап 5: Является ли использование исследуемой криптосистемы экономически выгодным в данном контексте?



**Рисунок 2 - Процесс оценки эффективности криптографической защиты**

Этапы 1-3 направлены на моделирование угроз, которым подвергаются информационные ресурсы, защищаемые с использованием исследуемой криптосистемы. Первый этап - определение объекта исследования. Здесь описываются конкретные характеристики криптосистемы. На втором этапе задаются параметры, определяющие тип потенциальных взломщиков криптосистемы. Между характеристиками криптосистемы и возможными типами атак существует связь. Аналогично, навыки, уровень доступа, бюджет в распоряжении злоумышленника определяют типы атак, которые он может предпринять [7]. Таким образом, при наличии формальных

представлений исследуемой криптосистемы и потенциальных злоумышленников мы можем перейти к этапу 3 на рис.2, т.е. определить типы атак, которым подвержена криптосистема, а также вероятность их реализации.

Этап 4 представляет собой анализ устойчивости криптосистемы к атакам, определенным на этапе 3. Необходимо обеспечить специалиста набором инструментальных средств для проведения исследований. Программный комплекс [14, 15], описанный в [6], предназначен для оценки стойкости асимметричных криптосистем к современным методам криптоанализа. Цель этапа 4 – оценить риск нарушения безопасности информационных ресурсов, защищенных с использованием исследуемой криптосистемы. Наконец, этап 5 предполагает использование различных подходов к оценке экономической эффективности инвестиций в СКЗИ на основании данных, полученных на этапах 1-4.

#### **4. Математическая модель угроз безопасности информационных ресурсов**

Задача оценки криптографической защищенности информационных ресурсов сводится к выделению подмножества атак, которым может подвергаться криптосистема в данном контексте использования, и определению устойчивости системы к этим атакам. Под устойчивостью системы будем понимать ее *криптостойкость*, т.е. способность противостоять атакам криптоаналитика [16].

Определение криптостойкости является научной задачей, для решения которой необходим набор инструментальных средств, позволяющих моделировать поведение злоумышленника при попытках взлома системы с использованием различных методов криптоанализа (см., например, [17]). Описание реализации программного комплекса, предназначенного для исследования стойкости асимметричных криптосистем с использованием математических методов, выходит за рамки данной статьи и приведено в [6].

Для определения эффективности криптосистемы имеет смысл проверять ее устойчивость не ко всем возможным атакам, а к тем, которые представляют для нее наибольшую угрозу. Состав множества потенциально опасных атак зависит от типа криптосистемы и условий использования криптосистемы.

Для выделения набора атак, которым подвержена криптосистема, построим математическую модель угроз безопасности защищаемых информационных ресурсов, основываясь на следующих предположениях:

- Один взломщик может предпринять атаки различного типа, а одна и та же атака может исходить от разных взломщиков;
- К одной и той же криптосистеме применимы атаки различного типа, а одна и та же атака позволяет взломать различные криптосистемы;
- Злоумышленник с наибольшей вероятностью выберет ту атаку, которая обеспечит максимальный результат при фиксированных затратах, либо наименее затратный вариант из множества атак, приводящих к одинаковому результату.

Пусть  $A \subseteq A_1 \times A_2 \times \dots \times A_8$  - множество параметрических моделей атак, где

$A_j$  ( $j = \overline{1, 8}$ ) - множество значений  $j$ -го параметра модели атаки,

определяющего тип атаки в соответствии с критериями разработанной классификации (см. [18]):

$A_1$  - по доступу к открытому коду;

$A_2$  - по контролю над процессом шифрования;

$A_3$  - по исходу атаки;

$A_4$  - по объему необходимых ресурсов;

$A_5$  - по степени применимости к различным шифрам;

$A_6$  - по используемым средствам;

$A_7$  - по последствиям атаки;

$A_8$  - по возможности распараллеливания.

Каждая модель  $\vec{a} \in A$  представляет собой вектор  $(a_1, a_2, \dots, a_8)$ , где

$a_j \in A_j$ ,  $j = \overline{1, 8}$ . Заметим, что, поскольку множества значений

параметров модели атаки конечны, то мощность множества моделей атак

$$|A| \leq \prod_{j=1}^8 |A_j|.$$

Пусть  $B \subseteq B_1 \times B_2 \times \dots \times B_6$  - множество параметрических моделей злоумышленников, где  $B_j$  ( $j = \overline{1, 6}$ ) - множество значений  $j$ -го параметра модели злоумышленника в соответствии с критериями описанной в [18] классификации:

$B_1$  - по технической оснащенности;

$B_2$  - по конечной цели;

$B_3$  - по доступу к шифрующим средствам;

$B_4$  - по уровню подготовки;

$B_5$  - по первичной информации о средстве шифрования;

$B_6$  - по возможности кооперации.

Каждая модель  $\vec{b} \in B$  задана в виде вектора  $(b_1, b_2, \dots, b_6)$ , где

$$b_j \in B_j, j = \overline{1, 6}, |B| \leq \prod_{j=1}^6 |B_j|.$$

Пусть  $C \subseteq C_1 \times C_2 \times \dots \times C_5$  - множество параметрических моделей криптосистем, где  $C_j$  ( $j = \overline{1, 5}$ ) - множество значений  $j$ -го параметра модели криптосистемы в соответствии с многокритериальной классификацией, описанной в [18]:

$\tilde{N}_1$  - по доступности информации о криптоалгоритме;

$\tilde{N}_2$  - по количеству ключей;

$\tilde{N}_3$  - по стойкости криптоалгоритма;

$\tilde{N}_4$  - по используемым средствам;

$\tilde{N}_5$  - по наличию сертификата;

Каждая модель  $\vec{c} \in C$  представляет собой вектор  $(c_1, c_2, \dots, c_5)$ , где

$c_j \in C_j, j = \overline{1, 5}$ ; мощность множества моделей криптосистем

$$|C| \leq \prod_{j=1}^5 |C_j|.$$

При дальнейшем изложении для краткости слово «модель» применительно к модели атаки, модели злоумышленника и модели криптосистемы будем опускать.

С каждой атакой будем связывать значение риска, вычисляемое по общеизвестной формуле на основе двух факторов – вероятности происшествия и тяжести возможных последствий:

$$\text{Риск} = \text{Влияние} \cdot \text{Вероятность}$$

Обозначим через  $\mathfrak{R} : A \times B \times C \rightarrow [0; 1]$  функцию, задающую уровень

риска, связанного с атакой  $\vec{a} \in A$  в условиях, когда она может быть

применена злоумышленником  $\vec{b} \in B$  для взлома криптосистемы  $\vec{c} \in C$ .

Пусть  $I : C \times A \rightarrow [0; 1]$  - функция влияния (от англ. *impact* – влияние, воздействие). Под влиянием мы будем понимать степень ущерба от применения атаки  $\vec{a} \in A$  к криптосистеме  $\vec{c} \in C$ .

Пусть  $P : B \times A \rightarrow [0; 1]$  - вероятность того, что злоумышленник  $\vec{b} \in B$  предпримет атаку  $\vec{a} \in A$ , т.е. обладает ресурсами для ее осуществления и сочтет эту атаку целесообразной.

Тогда функция риска  $\mathcal{R}$  выражается следующим образом:

$$\mathcal{R}(\vec{a}, \vec{b}, \vec{c}) = I(\vec{c}, \vec{a}) \cdot P(\vec{b}, \vec{a})$$

Определим функцию  $I(\vec{c}, \vec{a})$ . Для этого рассмотрим семейство функций

$I_{gh} : C_g \times A_h \rightarrow \mathbb{R}_+$ ,  $g = \overline{1, 5}$ ,  $h = \overline{1, 8}$ , где  $\mathbb{R}_+$  - множество неотрицательных действительных чисел. Здесь функция  $I_{gh}$  задает уровень взаимного влияния параметра криптосистемы  $c_g$  и параметра атаки  $a_h$ :

- $I_{gh}(c, a) = 0$ , если атака со значением параметра  $a \in A_h$  не применима к криптосистеме со значением параметра  $c \in C_g$ ;
- $0 < I_{gh}(c, a) < 1$ , если значение параметра криптосистемы  $c \in C_g$  снижает вероятность успешного применения атаки со значением параметра  $a \in A_h$ ;
- $I_{gh}(c, a) = 1$ , если значение параметра криптосистемы  $c \in C_g$  не влияет на применимость атаки с параметром  $a \in A_h$ ;
- $I_{gh}(c, a) > 1$ , если значение параметра криптосистемы  $c \in C_g$

указывает на то, что атака с параметром  $a \in A_h$  применима для ее взлома.

Уровень взаимного влияния параметров криптосистемы и атаки определяется на основе экспертных оценок.

Обозначим через  $\overline{I_{gh}} : C_g \times A_h \rightarrow [0; 1]$  нормированную функцию:

$$\overline{I_{gh}}(c, a) = \frac{I_{gh}(c, a)}{\sum_{\xi \in C_g} I_{gh}(\xi, a)}$$

Тогда уровень ущерба от применения атаки  $\vec{a} \in A$  к криптосистеме  $\vec{c} \in C$  вычисляется по следующей формуле:

$$I(\vec{c}, \vec{a}) = \min_{h=1,8} \prod_{g=1,5} \overline{I_{gh}}(c_g, a_h)$$

где атака и криптосистема заданы параметрами  $(a_1, a_2, \dots, a_8)$  и  $(c_1, c_2, \dots, c_5)$  соответственно. Заметим, что уровень влияния всех параметров криптосистемы на применимость атаки с заданным значением  $h$ -го параметра в этой формуле вычисляется по мультипликативному критерию:  $\prod_{g=1}^5 \overline{I_{gh}}(c_g, a_h)$ . Если значение хотя бы одного из параметров криптосистемы противоречит возможности применения атаки, то результатом оценки применимости атаки к криптосистеме будет нулевое значение, что соответствует нулевому уровню ущерба от атаки.

Определим функцию  $P(\vec{b}, \vec{a})$ . Для этого рассмотрим семейство функций

$P_{th}: B_t \times A_h \rightarrow \mathbb{R}_+$ ,  $t = \overline{1,6}$ ,  $h = \overline{1,8}$ . Здесь функция  $P_{th}$  задает уровень взаимного влияния параметра злоумышленника  $b_t$  и параметра атаки  $a_h$ :

- $P_{th}(b, a) = 0$ , если злоумышленник со значением параметра  $b \in B_t$  ни при каких обстоятельствах не будет использовать атаку со значением параметра  $a \in A_h$ ;
- $0 < P_{th}(b, a) < 1$ , если значение параметра злоумышленника  $b \in B_t$  снижает вероятность использования атаки со значением параметра  $a \in A_h$ ;
- $P_{th}(b, a) = 1$ , если значение параметра злоумышленника  $b \in B_t$  не влияет на вероятность использования атаки со значением параметра  $a \in A_h$ ;

- $P_{th}(b, a) > 1$ , если злоумышленник со значением параметра  $b \in B_t$  с большой вероятностью будет использовать атаку со значением параметра  $a \in A_h$ .

Уровень взаимного влияния параметров злоумышленника и атаки также определяется экспертами.

Обозначим через  $\overline{P_{th}} : B_t \times A_h \rightarrow [0; 1]$  нормированную функцию:

$$\overline{P_{th}}(b, a) = \frac{P_{th}(b, a)}{\sum_{\beta \in B_t} P_{th}(\beta, a)}$$

Тогда вероятность того, что злоумышленник  $\vec{b} \in B$  предпримет атаку  $\vec{a} \in A$ , вычислим по формуле:

$$P(\vec{a}, \vec{b}) = \min_{h=1,8} \prod_{t=1,6} \overline{P_{th}}(b_t, a_h)$$

где атака и злоумышленник заданы параметрами  $(a_1, a_2, \dots, a_8)$  и  $(b_1, b_2, \dots, b_6)$  соответственно.

Таким образом, общая формула для определения уровня риска, связанного с применением атаки  $\vec{a} \in A$  в условиях, когда эта атака может быть применена злоумышленником  $\vec{b} \in B$  для взлома криптосистемы  $\vec{c} \in C$ , имеет вид:

$$\begin{aligned} \Re(\vec{a}, \vec{b}, \vec{c}) &= \\ &= \min_{h=1,8} \prod_{g=1,5} \overline{I_{gh}}(c_g, a_h) \cdot \min_{h=1,8} \prod_{t=1,6} \overline{P_{th}}(b_t, a_h). \end{aligned}$$

Будем считать, что криптосистема  $\vec{c} \in C$  подвержена атаке  $\vec{a} \in A$  в условиях, когда ей угрожает злоумышленник  $\vec{b} \in B$ , если  $\Re(\vec{a}, \vec{b}, \vec{c}) > \theta$ , т.е. связанный с ней уровень риска превышает заданное пороговое значение  $\theta$ , где  $\theta \in [0; 1]$ . Допустимый уровень риска



$\theta$  является настраиваемым параметром модели угроз криптосистемы. Значение  $\theta$  задается с учетом двух критериев:

- критичности защищаемых данных;
- временных и других ресурсов, доступных специалисту, который осуществляет аудит системы.

В общем случае:

- Криптосистема может включать несколько подсистем (например, генератор ключей и симметричный шифратор), к каждой из которых применим свой набор атак;

- На криптосистему может нападать несколько злоумышленников. Множество атак, которым подвержена криптосистема, состоящая из

подсистем  $\vec{c} \in C'$  ( $C' \subseteq C$ ), в условиях, когда ей угрожают

злоумышленники  $\vec{b} \in B'$  ( $B' \subseteq B$ ), будем определять по формуле

$$\Lambda = \bigcup_{\vec{b} \in B'} \bigcup_{\vec{c} \in C'} \lambda(\vec{b}, \vec{c}), \quad \text{где}$$

$$\lambda(\vec{b}, \vec{c}) = \left\{ \vec{a} \in A : \Re(\vec{a}, \vec{b}, \vec{c}) > \theta \right\} \quad \text{при заданном уровне}$$

риска. Для оценки защищенности криптосистемы необходимо с использованием инструментальных средств оценить ее способность противостоять атакам, входящим в множество  $\Lambda$ .

В описанной математической модели сделаны следующие допущения:

- Не учитывается зависимость параметров атаки от сочетания параметров криптосистемы, хотя влияние каждого параметра принимается во внимание;

- Не учитывается возможность совместных действий со стороны взломщиков различных типов, хотя можно задать модель нападения со стороны однородного коллектива злоумышленников.

Исправление модели с учетом указанных допущений привело бы к ее значительному усложнению. Вопрос о том, насколько эти допущения снижают точность моделирования угроз безопасности, подлежит дальнейшим исследованиям.

На данный момент обнаружены две проблемы, связанные с практической реализацией разработанной модели в виде программного инструментария для аудитора:

- Получение экспертных оценок взаимного влияния параметров криптосистемы и атаки, а также злоумышленника и атаки;

- Поддержание базы оценок в актуальном состоянии, т.к.
  - с ростом вычислительных мощностей, изменением цен на аппаратные и программные средства и под влиянием других факторов уровень взаимного влияния параметров может меняться;
  - с появлением новых видов атак может возникнуть необходимость дополнения разработанных классификационных схем новыми критериями, что потребует введения новых зависимостей для соответствующих параметров моделей.

Методика оценки	Преимущества	Недостатки
Коэффициент возврата инвестиций	<ul style="list-style-type: none"> <li>Показатель, понятный финансистам</li> </ul>	<ul style="list-style-type: none"> <li>Отсутствие достоверных методов расчета в области ИТ</li> <li>«Статичный» показатель</li> </ul>
Совокупная стоимость владения	<ul style="list-style-type: none"> <li>Позволяет оценить целесообразность реализации проекта на основании оценки только затрат</li> <li>Предполагает оценку затрат на различных этапах всего жизненного цикла системы</li> </ul>	<ul style="list-style-type: none"> <li>Не учитывает качество системы безопасности</li> <li>«Статичный» показатель</li> <li>Показатель, специфичный для ИТ</li> </ul>
Дисконтированные показатели эффективности инвестиций	<ul style="list-style-type: none"> <li>Показатель, понятный финансистам</li> <li>Учитывает зависимость потока денежных средств от времени</li> <li>Учитывает все потоки денежных средств, связанные с реализацией проекта</li> </ul>	<ul style="list-style-type: none"> <li>Сложность расчета</li> </ul>

**Таблица 1 - Сравнительный анализ методов оценки эффективности инвестиций в средства обеспечения ИБ**

## 5. Расчет эффективности капитальных вложений в использование СКЗИ

На основании анализа преимуществ и недостатков методом оценки эффективности инвестиций в средства обеспечения ИБ (см. табл.1) был сделан вывод, что оптимальным является *метод дисконтированных показателей* [19], позволяющий получить наиболее полное представление о целесообразности капитальных вложений, хотя и требующий много времени и усилий на расчет экономических показателей.

Определим денежные потоки, связанные с использованием СКЗИ, за период  $t$  (где  $t = 0, 1, 2, \dots, T$  - периоды,  $T$  – горизонт расчета).

Затраты  $Cost_t$  на приобретение, установку и эксплуатацию СКЗИ могут быть определены очень точно, т.к. основной объем затрат составляет оплата труда персонала службы безопасности.

С защищаемой информацией связаны значения дохода  $Profit_t$  и ущерба  $Loss_t$  от НСД к защищаемой информации в течение указанного промежутка времени  $t$ . Пусть результаты оценки способности криптосистемы противостоять атакам, представляющих для нее угрозу (см. п. 4) с использованием инструментальных средств показали, что с вероятностью  $R_t$  в  $t$ -м периоде злоумышленник получит доступ к защищаемой информации. Тогда математическое ожидание дохода  $R_t$ , связанного с использованием оцениваемой СКЗИ, вычисляется по формуле:

$$R_t = -Cost_t + Profit_t \cdot (1 - P_t) - Loss_t \cdot P_t$$

На основании этих данных о притоках и оттоках денежных средств вычисляются финансово-экономические показатели эффективности инвестиций в криптосистему и делаются выводы о ее соответствии потребностям организации.

## **6. Анализ разработанной методики**

К достоинствам разработанной методики оценки эффективности криптосистем можно отнести следующее:

- В основе методики лежит комплексный подход к оценке рисков, основанный на формализованном пятиэтапном процессе оценки эффективности криптосистемы и математической модели угроз.
- Методика является универсальной и подходит для организаций различного масштаба как правительственного, так и коммерческого сектора.
- Методика помогает провести анализ рисков, сочетающий количественные и качественные методы анализа, и сделать обоснованный выбор мер и средств криптографической защиты.
- Методика позволяет провести экономическое обоснование расходов организации на обеспечение информационной безопасности и непрерывности бизнеса с использованием СКЗИ.
- Методика позволяет оценить не только риски, связанные с защищаемыми информационными ресурсами, но и выгоду, которую может принести внедрение СКЗИ.

К недостаткам разработанной методики можно отнести следующее:

- Использование методики требует специальной подготовки и высокой квалификации аудитора.
- Методика в большей степени подходит для аудита уже существующих криптосистем, находящихся на стадии эксплуатации, чем для криптосистем, находящихся на стадии разработки.
- Аудит с использованием разработанной методики - достаточно трудоемкий процесс, который может потребовать месяцев работы специалиста.
- Использование методики предполагает наличие экспертов, способных дать достоверные оценки объема потерь от реализации угроз ИБ;
- На данный момент программное обеспечение, автоматизирующее процесс построения модели угроз на основе данных об особенностях реализации исследуемой криптосистемы и типах потенциальных злоумышленников, находится на стадии разработки.

## **7. Список литературы**

- [1] ГОСТ Р 50922-96. Защита информации. Основные термины и определения.
- [2] Яковлев А.В., Безбогов А.А., Родин В.В., Шамкин В.Н. Криптографическая защита информации: учебное пособие / Тамбов: Изд-во Тамб. гос. техн. ун-та, 2006. – 140 с.
- [3] Иванов В.П. Математическая оценка защищенности информации от несанкционированного доступа // "Специальная техника". 2004, N 1. -с. 58-64.
- [4] Brassard J. Modern Cryptology. Springer-Verlag, Berlin - Heidelberg, 1988. - 107 p. (Русский перевод: Брассар Ж. Современная криптология. Полимед, 1999. 176 с.)
- [5] Kerckhoffs A. La cryptographie militaire // Journal des sciences militaires, vol. IX. P. 5-38, Jan. 1883, (P. 161-191, Feb. 1883).

[6] Savelieva A. Formal methods and tools for evaluating cryptographic systems security // St. Petersburg, ISP RAS, In Proceedings of the Second Spring Young Researchers Colloquium on Software Engineering (SYRCoSE'2008), 2008, Vol 1. P. 33-36.

[7] Schneier B. Modeling security threats // Dr. Dobb's Journal, December, 1999.

[8] Баричев С.Г. Основной вопрос криптографии // Chief Inforamtion Officer – руководитель информационной службы. #5 (37), 2005, с. 93-95.

[9] Куканова Н. Методы и средства анализа рисков и управление ими в ИС // Byte/Россия. - 2005. - № 12. - С. 69-73.

[10] Куканова Н. Современные методы и средства анализа и управления рисками информационных систем компаний // Опубликовано: [http://www.dsec.ru/about/articles/ar\\_compare/](http://www.dsec.ru/about/articles/ar_compare/) 2006.03.17

[11] CRAMM V Official website // Siemens Enterprise Communications Limited 2006. Available at: [www.cramm.com](http://www.cramm.com)

[12] RiskWatch Official website // RiskWatch, Inc. Available at: <http://www.riskwatch.com/>

[13] Digital Security: ГРИФ. Система анализа и управления информационными рисками // Digital Security, 2002-2008. Опубликовано: <http://www.dsec.ru/products/grif/>

[14] Авдошин С.М., Савельева А.А. Инструментальные средства криптоанализа асимметричных шифров. - М.: ВНИИЦ, 2008. - №50200800603.

[15] Авдошин С.М., Савельева А.А. Инструментальные средства криптоанализа асимметричных шифров. Свидетельство о государственной регистрации в Реестре программ для ЭВМ № 2005612258 от 22.05.08.

[16] Ростовцев А.Г., Михайлова Н.В. Методы криптоанализа классических шифров // 1998. Опубликовано: <http://crypto.hotbox.ru/download/cryptoan.zip>

[17] Авдошин С.М., Савельева А.А. Криптоанализ: современное состояние и перспективы развития. Новые технологии; М.: Машиностроение, 2007. - 24 с. - (Библиотечка журнала "Информационные технологии"; Приложение к журналу "Информационные технологии"; № 3)

[18] Авдошин С.М., Савельева А.А. Проблемы оценки криптозащищенности информационных систем // «Новые информационные технологии». Тезисы докладов XVI Международной студенческой школы-семинара - М.: МИЭМ, 2008. С. 15-29.

[19] Старик Д.Э. Расчеты эффективности инвестиционных проектов. М.: Финстатинформ, 2001.

# Software Portability: Forty Years Later

**Alexey Khoroshilov**  
**Institute for System**  
**Programming**  
**Russian Academy of Sciences**  
**email: khoroshilov@ispras.ru**

**Denis Silakov**  
**Institute for System Programming**  
**Russian Academy of Sciences**  
**email: silakov@ispras.ru**

## Abstract

The software portability problem is a well-known problem in software engineering, and there are many approaches to fix it. Nevertheless, when the portability question is not considered in time and unpleasant consequences happen as a result, these situations perpetually occur.

This report draws attention to this problem. We hope to persuade software developers and their customers to more carefully consider the portability requirement at the initial stage of a project. This report discusses several examples of incorrect solutions taken. We consider the most popular approaches to fix the portability problem and discuss their virtues and shortcomings.

**Keywords:** Portability; interoperability.



# Проблема переносимости приложений – сорок лет спустя

**Денис Силаков**  
**Институт системного**  
**программирования**  
**Российская Академия Наук**  
**email: silakov@ispras.ru**

**Алексей Хорошилов**  
**Институт системного**  
**программирования**  
**Российская Академия Наук**  
**email: khoroshilov@ispras.ru**

## Тезисы

Проблема переносимости приложений между программно-аппаратными платформами имеет длинную историю. За это время появилось множество подходов к её решению. Тем не менее, постоянно возникают ситуации, когда выясняется, что данному вопросу вовремя не было уделено должное внимание и это привело к неприятным последствиям.

Настоящий доклад ставит своей целью привлечь внимание к вопросу переносимости программного обеспечения и добиться, чтобы и заказчики, и разработчики ПО более серьезно подходили к рассмотрению требования переносимости на начальных этапах проекта. Мы представим примеры последствий недальновидных решений этого вопроса, рассмотрим наиболее распространенные подходы к обеспечению переносимости ПО, а также обсудим их достоинства и недостатки.

**Keywords:** Переносимость; мобильность; интероперабельность.

## 1. Введение

Проблема переносимости приложений между различными программно-аппаратными платформами ненамного моложе собственно компьютерных программ. Еще в конце шестидесятых годов озабоченность некоторых сотрудников AT&T Labs проблемой переносимости ОС UNIX на новые аппаратные платформы привела к созданию языка Си; однако темпы развития компьютерной индустрии таковы, что проблемы сорокалетней давности кажутся достаточно простыми и решаемыми, по сравнению с тем, что мы имеем сегодня.

Стремительное развитие связанных с компьютерами отраслей приводит к постоянному появлению новых программно-аппаратных платформ, информационных систем, и т.п., в то время как устаревшие комплексы уходят в небытие.

Производители ПО, как правило, заинтересованы в быстром переносе своих продуктов на новые системы, чтобы захватить соответствующую долю рынка. Если приложение изначально проектировалось с оглядкой на возможность портирования, то этот процесс может оказаться существенно дешевле создания нового продукта. Будет проще и конечным пользователям, которые в новой системе увидят то же самое приложение, с которым работали раньше, что также способствует популярности продукта.

Исчезновение же с рынка тех или иных платформ приводит к появлению унаследованного ПО — программных продуктов, необходимых для функционирования той или иной организации, но требующих для работы устаревшей программно-аппаратной платформы. В случае выхода из строя аппаратного обеспечения может оказаться, что найти ему замену очень сложно, дорого, а иногда и попросту невозможно, так как устаревшая ОС не работает на современном оборудовании ввиду отсутствия необходимых драйверов. Для многих предприятий задача переноса таких приложений в более современное окружение является крайне актуальной [1,2].

## **2. Примеры из современности**

Несмотря на то, что о проблеме переносимости известно достаточно давно и ее решению посвящено множество работ и исследований, постоянно возникают ситуации, когда выясняется, что данному вопросу вовремя не было уделено должное внимание и это привело к неприятным последствиям. Рассмотрим несколько реальных примеров таких ситуаций.

Некоторая американская компания разрабатывала ряд программных продуктов для платформы Microsoft Windows. Не так давно руководство компании осознало, что активно растущий рынок пользователей MacOS в США также представляет для них лакомый кусочек, в особенности,

учитывая большую заинтересованность пользователей в их продуктах.

Руководство инициировало анализ возможности выпуска продуктов для MacOS, который выявил следующую картину. Ведущий разработчик одной из линеек продуктов, имевший опыт разработки переносимых приложений, по собственной инициативе организовал внутреннюю архитектуру этих приложений с четким выделением платформенно-зависимой функциональности. В других приложениях компании собственная функциональность оказалась тесно завязана на особенности целевой операционной системы. В результате затраты по доработке, тестированию и выводу на рынок первого продукта были признаны окупаемыми в краткосрочной перспективе. А вот экономическая целесообразность портирования других продуктов была поставлена под сомнение, ввиду необходимости практически полного переписывания их кода. Таким образом, невнимание к проблеме переносимости обернулось потерей части прибыли и проблемам с захватом позиций на перспективном рынке.

Другой пример недальновидного подхода к этому вопросу проявился в ходе организации проекта по разработке пакета свободного программного обеспечения (СПО) для образовательных учреждений России. Практически все программное обеспечение, которое разрабатывалось по заказу Министерства образования РФ в последние годы было предназначено для работы исключительно на платформе Microsoft Windows. Поэтому при внедрении пакета СПО на основе операционной системы Linux большая часть разработанных ранее образовательных программ оказалась недоступна, и только часть из них удавалось запустить с помощью эмулятора wine [3].

Схожие проблемы возникали и в стане коммерческих заказчиков. Известно несколько случаев, когда при разработке интернет-сервисов заказчик ограничивался требованием совместимости с браузером Internet Explorer, а через некоторое время под давлением клиентов был вынужден дорабатывать ПО для поддержки Mozilla Firefox.

Приведенные примеры демонстрируют, как невнимание к проблеме переносимости может привести к различным негативным техническим, экономическим и политическим последствиям:

- проблемам с поддержкой ПО в долгосрочной перспективе;
- сокращению доступных рынков и недополучению прибыли;
- попаданию в зависимость от одного поставщика.

Как мы уже отметили, проблема известна давно, и за время эволюции программного обеспечения появилось достаточно много подходов к созданию приложений, переносимых между различными системами; рассмотрим те, что представляются наиболее популярными.

### **3. Бинарная совместимость**

Перенос приложения на новую программно-аппаратную платформу может пройти безболезненно для разработчиков, если старая и новая системы совместимы на бинарном уровне, то есть новая система позволяет использовать старые двоичные файлы приложения без каких-либо изменений.

Для реализации такой возможности целевая платформа должна поддерживать соответствующий формат исполняемых файлов и файлов разделяемых библиотек, а также обладать совместимостью на уровне двоичного интерфейса приложений (Application Binary Interface, ABI).

Одной из важных составляющих бинарной совместимости является набор функций, предоставляемых системой. Полной идентичности таких наборов ожидать трудно, однако во многих случаях пересечение множеств предоставляемых функций является достаточно большим. Многие производители операционных систем в настоящее время заботятся об обратной совместимости своих продуктов на бинарном уровне — гарантируется, что приложение, работающее в некоторой версии ОС, будет работать без перекомпиляции в более новых версиях системы.

Свойство бинарной совместимости жестко связано с аппаратной частью платформы — в силу технических причин в

общем случае сложно достичь совместимости систем, работающих на различных архитектурах. В некоторых конкретных ситуациях операционные системы предоставляют возможность запуска приложений, написанных для той же системы, но на другой платформе — так, операционная система MacOS X, работающая на компьютерах Apple с процессорами Intel, использует динамический транслятор Rosetta для выполнения программ, предназначенных для машин с процессорами PowerPC [4]. Однако пользоваться такой возможностью следует с осторожностью. Во многих случаях совместимость обеспечивается за счет некоторого дополнительного слоя совместимости между системой и приложением, который может и не гарантировать полной совместимости — так, уже упомянутая Rosetta позволяет исполнять код для процессоров G3, G4 и Altivec, но не для G5. Кроме того, дополнительный компонент системы является дополнительным потенциальным источником ошибок, а использование посредника в общем случае снижает производительность.

#### **4. Переносимость исходного кода**

К сожалению, многие существующие платформы не совместимы на бинарном уровне и не способны загружать исполняемые файлы друг друга (хотя бы потому, что используют различные форматы файлов, соглашения о вызове функций и т.п.). Альтернативой использованию одних и тех же бинарных файлов явилось использование одного и того же исходного кода для сборки приложения на различных системах.

В те времена, когда программы писались на ассемблере конкретной аппаратной платформы, добиться компиляции приложения на системе с другой архитектурой было практически нереально. Существенной подвижкой в этом направлении стало создание высокоуровневых языков программирования, не привязанных к конкретной архитектуре и позволяющих использовать одни и те же конструкции на различных системах.

Для обеспечения такой возможности целевые системы должны предоставлять компиляторы для используемого языка программирования. Естественно, что компиляторы для

различных систем создаются различными производителями и могут достаточно сильно отличаться друг от друга.

Частично эту проблему решают международные стандарты, существующие для многих языков программирования. Однако далеко не все компиляторы поддерживают стандарты в полном объеме (хотя, как правило, не поддерживаются некоторые специфические конструкции языка, в плане же предоставления библиотечных функций ситуация гораздо лучше). Другой проблемой является относительная узость стандартов — несмотря на наличие во многих из них перечня функций, которые должны предоставляться любой удовлетворяющей стандарту средой разработки, эти перечни не описывают существенную часть функциональности, которая могла бы быть полезна при создании приложений — например, функции графического интерфейса.

Помимо стандартов языков программирования, существуют стандарты, описывающие интерфейс прикладных программ (API, Application Programming Interface) — например, POSIX. Однако такие стандарты также, достаточно узки, и являются недостаточными для написания большинства приложений.

Для реализации функциональности, не охваченной никакими стандартами, полезными могут оказаться кросс-платформенные библиотеки, предоставляющие необходимый набор функций — например, существует ряд библиотек для создания графического интерфейса, которые можно использовать как в Windows, так и в Linux.

## **5. Интерпретируемые языки**

Еще одним шагом по обеспечению переносимости приложений явилось использование интерпретируемых языков — т.е. языков, использование которых не подразумевает создания исполняемых файлов в формате целевой операционной системы. Вместо этого интерпретатор последовательно считывает и выполняет инструкции непосредственно из текста программы.

Однако прямолинейная интерпретация достаточно неэффективна — у интерпретатора практически нет возможностей для оптимизации кода. Для повышения

эффективности во многих языках (Java, Perl, семейство .NET) исходный код сначала транслируется в некоторое промежуточное представление (байт-код), который уже подается на вход интерпретатору.

Еще большей производительности удастся достигнуть при использовании компиляции “на лету” (Just In Time compilation), когда байт-код транслируется в машинный код во время работы программы. Однако разработчикам JIT-компиляторов также приходится идти на компромисс между временем, уходящим на оптимизацию, и эффективностью получаемого кода, в результате чего производительность приложений может уступать коду, создаваемому “обычными” компиляторами. Кроме того, использование данной технологии увеличивает потребление памяти приложением, поскольку оттранслированный код также хранится в оперативной памяти. Также следует иметь в виду, что эффективность реализации интерпретаторов, также как и JIT-компиляторов, на различных платформах может отличаться.

## **6. Эмуляция ABI других систем**

Говоря о бинарной переносимости, мы уже упомянули, что в ряде случаев операционная система может обеспечивать бинарную совместимость с другой системой за счет дополнительного слоя совместимости. Возможности самих систем в этом плане достаточно ограничены — как правило, такой слой совместимости вводится для поддержки приложений той же самой системы, но для другой архитектуры.

В то же время существует много разработок, позволяющих загружать файлы других операционных систем путем использования транслятора, способного загружать файлы требуемого формата, преобразуя вызовы функций, осуществляемые внутри файла, в соответствующие вызовы текущей ОС (фактически, такой транслятор реализует ABI старой системы в новой системе).

В качестве примера можно привести wine [3], предназначенный для запуска Windows-приложений в Linux, а также sugwin [5], обеспечивающий переносимость в обратную сторону. При этом, например wine достаточно легко

использовать как часть приложения, не полагаясь на его доступность в целевой системе.

Недостатком использования такого рода эмуляторов является потенциальная неполнота реализации интерфейса, необходимого приложению. Так, разработчики того же wine ориентируются на публично доступную информацию об API Windows (например, стандарт ECMA-234); так что если приложение использует какие-то недокументированные возможности этой ОС, то попытка его запуска в wine может оказаться неудачной.

## **7. Веб-сервисы и сервисно-ориентированная архитектура**

Необходимость создания сложных распределенных систем привела к созданию парадигм, подразумевающих разбиение системы на отдельные компоненты, взаимодействующие друг с другом по строго определенным протоколам. При таком подходе каждый компонент может разрабатываться независимо, а процесс его переноса на другую платформу никак не затрагивает других частей системы.

К одной из первых попыток описания механизма взаимодействия компонентов распределенной системы можно отнести спецификацию CORBA (Common Object Request Broker Architecture), разработанную консорциумом OMG [6].

Однако CORBA в силу ряда причин не снискала большой популярности, и в настоящее время гораздо больший интерес проявляется к веб-сервисам, использующим протоколы обмена сообщениями на базе XML. При проектировании сложных распределенных программных комплексов используется парадигма сервисно-ориентированной архитектуры (Service-oriented architecture, SOA [7]); при этом программные комплексы часто реализуются как набор веб-сервисов.

## **8. Виртуализация**

В ряде случаев для запуска приложений необходимо аппаратное обеспечение, которое в силу ряда причин не доступно человеку, запускающему программу. В таком случае



на помощь приходят виртуальные машины, эмулирующие работу некоторой аппаратной платформы. На такой виртуальной машине устанавливается операционная система и другое окружение, необходимое приложению, а само приложение запускается уже в родной для него среде.

Возможность запуска приложения на виртуальной машине зависит, в основном, от возможностей самой машины, нежели от разработчиков приложения. Тем не менее, программы, работающие с аппаратурой напрямую, могут встретить определенные трудности, поскольку им будет предоставлен доступ к устройствам виртуальной машины, а не непосредственно компьютера. Такая особенность ограничивает, например, возможность работы с графическими ускорителями внутри виртуальных машин.

Использование виртуальной машины достаточно ресурсоемко — ведь помимо собственно приложения, ресурсы компьютера потребляются самой машиной, а также работающими внутри нее программами, необходимыми для функционирования приложения (например, операционной системой). Поэтому выигрыш в производительности достигается, как правило, только в случае запуска машин, эмулирующих достаточно маломощные платформы на более производительных системах.

## **9. Заключение**

Если невнимательность к проблеме переносимости приводит к негативным последствиям и существует множество путей по ее решению, то возникает вопрос: так почему же ИТ индустрия не переориентируется на разработку переносимого ПО. Несложно догадаться, что разработка переносимого ПО имеет свои недостатки.

Среди рассмотренных видов переносимости приложений очень привлекательным с точки зрения разработчиков является перенос непосредственно бинарных файлов на новую систему, позволяющий при относительно небольших затратах (в основном уходящих на тестирование) получить на новой системе приложение, имеющее всю необходимую

функциональность. При этом потери в производительности если и возникают, то совсем небольшие.

Однако для любой ОС число платформ, совместимых с ней на бинарном уровне, достаточно невелико. Использование эмуляторов может расширить их круг, однако эмулятор — дополнительный потенциальный источник ошибок, который при этом может и не предоставлять всех необходимых функций.

Потенциально больший охват дает переносимость исходного кода. Сложность портирования в этом случае может варьироваться в зависимости от того, насколько такая возможность учитывалась при разработке приложения; полезной с этой точки зрения является ориентация на различные интерфейсные стандарты, регламентирующие взаимодействие приложения с окружающей средой. Но существующие стандарты охватывают достаточно небольшую функциональность; в ряде случаев может помочь использование кросс-платформенных библиотек, другой же альтернативой является использование интерпретируемых языков.

Спецификации таких языков не привязаны к конкретной платформе и можно полагаться на то, что интерпретаторы на разных системах поддерживают один и тот же набор функций. Среди недостатков подхода можно выделить меньшую производительность по сравнению с бинарным кодом.

Архитектура SOA затрагивает более сложную проблему организации сложных программных комплексов, предлагая строить их в виде набора достаточно изолированных компонентов, каждый из которых может работать на своей собственной платформе и в случае необходимости может быть перенесен на другую (либо заменен на альтернативную реализацию).

Использование виртуальных машин также не требует больших усилий со стороны разработчиков ПО, хотя этот способ достаточно накладен, как в смысле производительности, так и ввиду необходимости иметь лицензии на все используемые операционные системы. Применение виртуализации оправдано в тех случаях, когда перенос

приложения каким-то другим способом представляется экономически неэффективным. В частности, это относится ко многим унаследованным системам, для которых портирование на новую платформу означало бы практически полное переписывание приложения.

В таблице 1 мы постарались отразить влияние различных методик, используемых для увеличения переносимости приложения, на процесс разработки, а также на характеристики получаемого продукта.

Ориентация на существующие стандарты ПО полезна в любом случае, однако не является панацеей от всех бед ввиду относительной узости (или даже отсутствию) стандартов во многих областях. Использование различных медиаторов (библиотек, интерпретаторов, дополнительных слоев совместимости), отделенных от самого приложения, расширяет число потенциальных целевых систем. Однако разработка медиаторов собственными силами ведет к увеличению затрат на производственный процесс; использование же сторонних продуктов ставит разработчика в зависимость от других поставщиков. То же самое верно и для эмуляторов и виртуальных машин — использование готовых решений достаточно дешево (фактически, при этом необходимо лишь дополнительное тестирование в новой системе плюс соответствующие лицензионные отчисления), однако ставит в зависимость от их производителей. Риски зависимости снижаются при использовании свободных компонентов с открытым кодом, так как перспективы развития этих компонентов становятся более управляемыми.

Подводя итоги, отметим, что возможны ситуации, когда отказ от обеспечения переносимости разрабатываемого ПО оправдан; например, с достаточно большой долей вероятности такой выбор будет выгоден в краткосрочной перспективе. Однако при создании продуктов, которые планируется поддерживать в течении достаточно длительного периода, обеспечение переносимости может оказаться одним из ключевых факторов успеха. Повышение мобильности приложения в общем случае всегда связано с увеличением расходов на его разработку; однако чем раньше об этой

проблеме задумаются разработчики и архитекторы, тем меньше будет стоимость переноса приложения на новую платформу. Поэтому хотелось бы подчеркнуть, что вопрос обеспечения переносимости следует рассматривать в самом начале проекта, на стадии проектирования и выбора технологий и инструментов, которые будут использованы при его реализации.

**Таблица 1. Сравнительная характеристика путей  
достижения переносимости**

	Ориентация на стандарты	Медиаторы сторонних разработчиков	Собственные медиаторы	Эмуляция	Виртуализация
Производительность	Не влияет	Возможно снижение	Возможно снижение	Снижается	Снижается
Удорожание разработки	Доп. затраты из-за узости стандартов	Нет (при наличии готовых решений)	Да, зависит от числа целевых систем	Практически нет (при наличии готовых решений)	Практически нет (при наличии готовых решений)
Увеличение сроков разработки	Доп. затраты из-за узости стандартов	Незначительно	Да, зависит от числа целевых систем	Практически нет	Нет (при наличии готовых решений)
Расширение рынка	В рамках охвата стандарта	На системы, охваченные медиаторами	В соответствии с собственными возможностями	На системы, содержащие эмулятор	На системы, содержащие виртуальную машину
Долгосрочная поддержка	Существенно облегчается	Зависит от поддержки медиаторов	Доп. затраты на поддержку медиаторов	Зависит от поддержки эмуляторов	Зависит от поддержки виртуальных машин

## 10. Литература

- [1] James D. Mooney. "Bringing Portability to the Software Process". Technical Report TR 97-1, Dept. of Statistics and Computer Science, West Virginia University, Morgantown WV, 1997.
- [2] Ian Sommerville. Software Engineering, 8th Edition, Addison Wesley, 2006.
- [3] Wine - Open Source implementation of the Windows API. <http://www.winehq.org/>
- [4] Apple Rosetta. <http://www.apple.com/rosetta/>
- [5] Cygwin - Linux-like environment for Windows. <http://www.cygwin.com/>
- [6] OMG CORBA 3.0. [http://www.omg.org/technology/documents/formal/corba\\_2.htm](http://www.omg.org/technology/documents/formal/corba_2.htm)
- [7] Ben Margolis, Joseph Sharpe. SOA for the Business Developer: Concepts, BPEL, and SCA. MC Press, 2007.

## **Software development statistical management approach realization in testing**

**Alexandrov Alexandr**  
**Luxoft Professional**  
**Moscow, Russia**  
**email:**

**aalexandrov@luxoft.com**

**Galay Anatoly**  
**Luxoft Professional**  
**Moscow, Russia**  
**email:agalay@luxoft.com**

**Milkova Yasna**  
**Luxoft Professional**  
**Moscow, Russia**  
**email:ymilkova@luxoft.com**

### **Abstract**

Procedure approach that is widely used in organizations allows to understand whether project team follow established process or not. But unfortunately this approach do not ensure problem identification on the early stage and do not allow as to be sure in high quality of product.

Following to all established procedures do not ensure base for software quality evaluation. Such approach depend on quality of procedure and quality of it's execution. So the main imperfection of this approach is in the fact that project managers do not have quantitative methods to evaluate quality of the product. The only one thing that is transparented for them is whether planned task were completes or not.

The approach described is based on quantitative management and help to resolve this problem. In the article testing process was choosen to demonstrate main principles of quantitative management approach realization that allow to establish quantitative goals, analyse processes and develop and implement corrective/preventive actions earlier.

**Keywords:** Quantitative management; statistical management; CMMI-DEV ver 1.2; testing.

## **Количественное управление процессом тестирования.**

**Александров  
Александр  
Люксффт Профешнл  
Москва, Россия  
email:  
aalexandrov@luxoft.com**

**Галай Анатолий  
Люксффт Профешнл  
Москва, Россия  
email:agalay@luxoft.com**

**Милькова Ясна  
Люксффт Профешнл  
Москва, Россия  
email:ymilkova@luxoft.com**

### **Тезисы**

Процедурный подход, широко применяемый на практике, позволяет убедиться в следовании или отклонениях от заранее запланированных процессов. Но, используя этот подход, далеко не всегда удастся заранее выявить проблемы или быть уверенным в достижении требуемого уровня качества разрабатываемой программной системы.

Иначе говоря, одно лишь выполнение набора процедур не обеспечивает основу для оценки качества разрабатываемой программной системы. Более того, такой подход в высшей степени зависит от качества процедуры и от качества ее выполнения. Основной недостаток процедурного подхода состоит в отсутствии у менеджеров проектов количественных способов, с помощью которых они могли бы оценить качество разрабатываемой программной системы, единственный видимый им фактор показывает, были ли выполнены запланированные задачи.

Описанный в докладе подход (рассмотренный на примере процесса тестирования), основан на количественном управлении данными и позволяет решить эту проблему, устанавливая количественные цели по качеству, анализировать процессы и своевременно разрабатывать и предпринять корректирующие и/или предупреждающие действия- причем это можно сделать на более ранней стадии, когда цена ошибки еще не так дорога.

**Keywords:** Кколичественное управление; статистическое управление; CMMI-DEV ver 1.2, тестирование.



## **1. Принципы количественного управления в модели CMMI**

### **1.1. Практики CMMI-DEV версии 1.2 по количественному управлению**

В презентации будут рассмотрены возможные практические шаги по внедрению практики статистического управления на примере процесса тестирования.

На рисунке приведена схема области усовершенствование “Управление проектом на основе количественных данных” по модели CMMI-DEV версии 1.2.

Таким образом, при внедрении практики статистического управления, организации нужно предпринять следующие основные шаги:

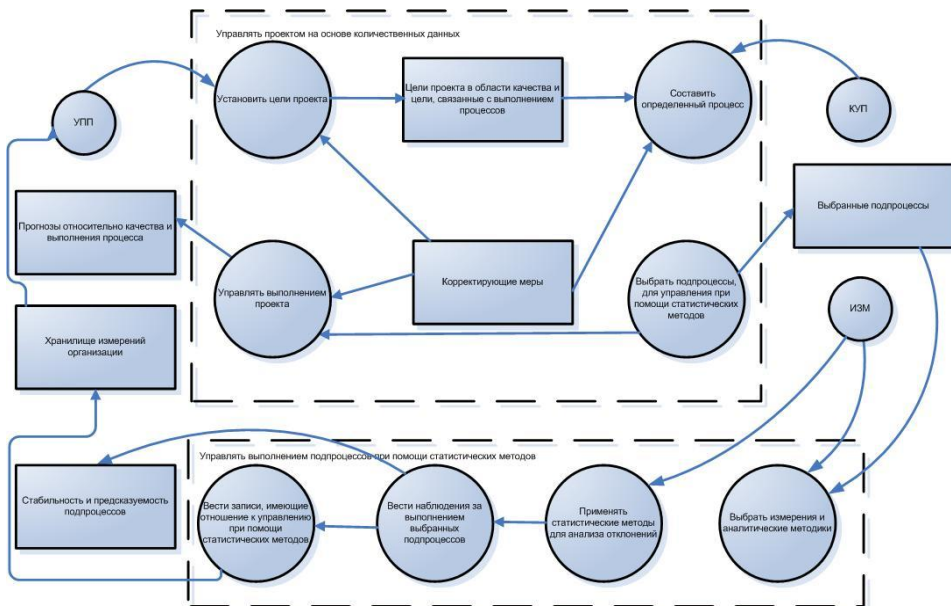
- определить, в какой информации нуждается компания (какие цели будут поставлены),
- определить метрики, отражающие достижение поставленных целей,
- определить, какие подпроцессы, влияющие на достижение поставленных целей, будут статистически управляться,
- определить, как будут протекать процессы статистического и количественного управления
- Выполнять процессы количественного и статистического управления.

Презентация будет в основном посвящена последним двум пунктам.

### **1.2. Некоторые определения и важные исходные позиции**

**Количественное управление** – это процесс использования данных проектных измерений, обработанных с помощью статистического управления подпроцессами (или каких-то других методов) для:

- определения того, обеспечат ли текущие значения



параметра процесса выполнение требований к нему в конце проекта,

- если текущие результаты не дают уверенности в выполнении конечных требований, то определение корректирующих действий, которые должны быть предприняты для обеспечения достижения установленных,
- Последующего контроля эффективности предпринятых мер.

**Статистическое управление** – это использование статистических методов для обработки и оценки результатов измерений параметров процессов в проекте. В результате применения таких методов становится возможным:

- определять границы, в которых значения параметра могут находиться, если подпроцесс выполняется

штатно (т.е. предсказывать значение параметра подпроцесса),

- определять значения контролируемого параметра, которые являются следствием воздействия каких-то особенных (единовременных) причин.

Важные замечания:

- параметры подпроцесса, выбираемого для статистического управления должны влиять на цель (одну из целей) проекта. Это даст возможность, контролируя статистически параметр подпроцесса обеспечивать управление достижением проектной цели,
- подпроцесс, выбираемый для статистического управления должен быть стабильным, т.е. иметь достаточно стабильные значения параметров, его характеризующих в случае выполнения данного подпроцесса по установленным правилам,
- важно, чтобы количество моментов времени для корректного измерения параметров процессов, подлежащих статистическому управлению, во время выполнения проекта было достаточно большим.

Одним из самых простых и чаще всего используемым методом статистического анализа являются Контрольные карты (Control Charts) различных типов. Наиболее универсальный тип контрольных карт, применимый практически в любых условиях, это XmR Chart.

Гистограмма полезна при определении степени приближения значений параметра к нормальному распределению.

## **2. Возможный сценарий применения количественного управления в тестировании**

### **2.1. Общие положения**

Под управлением процессом тестирования понимаются обоснованные действия, направленные на достижение целей тестирования, сформулированных в проекте. Роль таких целей могут играть, в частности, цели самого проекта.

Использование количественного управления тестированием позволяет:

- устанавливать в проекте количественные цели по качеству,

- анализировать (с применением статистических методов) ход процесса тестирования, направленный на достижение целей по качеству,
- обоснованно планировать и осуществлять предупреждающие и корректирующие действия, направленные на достижение целей качества, отслеживать эффект выполнения этих действий.

В процессе тестирования важнейшим параметром, который дает возможность получать метрики, характеризующие качество, как и процесса тестирования, так и самого продукта, является количество дефектов.

На основе этой простой метрики можно рассчитать, например, такие метрики, как:

- плотность дефектов, обнаруженных в ходе выполнения тестирования (software defect density-SDD),
- плотность дефектов обнаруженных в эксплуатации (post delivery defect density PDDD),
- коэффициент отвергнутых дефектов –(declined defect ratio- DDR).

## **2.2. Пример реализации статистического управления параметром SDD**

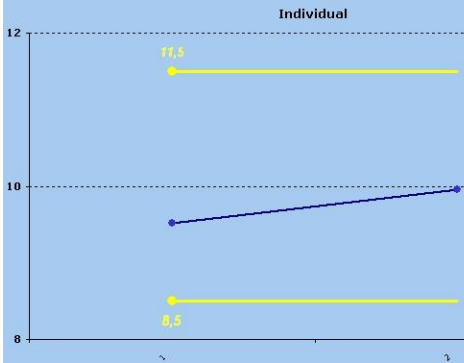
Попытаемся представить, каким образом можно количественно управлять подпроцессом тестирования. Управлять будем, например, на основе плотности дефектов, обнаруженных во время тестирования. Последовательность действий должна быть примерно следующей:

1. Установить каким-либо образом диапазон значений плотности дефектов, обнаруженных во время системного тестирования, попадание в который будет являться целью управления.
2. Определить моменты времени для контроля значений плотности дефектов
3. Определить, как будет определяться наличие особенного случая в зависимости от количества измерений или это правило не будет зависеть от количества измерений
4. В момент получения значения плотности дефектов с помощью XmR определить значения среднего и диапазон возможных отклонений.

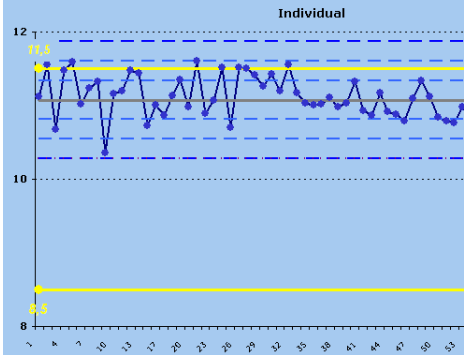
5. Сравнить полученные значения с установленной целью и при необходимости выработать корректирующие действия.

### **2.3. Возможные правила проведения статистического анализа параметров**

Возможные правила проведения статистического анализа параметров приведены в таблице.

Кол-во измерений	Причина (Обоснование)	Пример
Начальная фаза сбора значений параметра	Если количество доступных значений менее 3, то статистический анализ не проводится. Анализируются только значения параметров.	 <p>Ожидается попадание значения параметра в границы (желтые линии)</p>

Кол-во измерений	Причина (Обоснование)	Пример
Фаза накопления данных для статистической обработки	Если количество доступных значений в диапазоне <b>3-29</b> , считаем, что полностью доверять рассчитанным с помощью XmR chart статистическим параметрам нет оснований, но для вынесения суждения о наличии особенного случая можно использовать границы $\pm 2\sigma$	 <p>Значение, вышедшее за пределы <math>\text{Mean} \pm 2\sigma</math>, может считаться особенным.</p>
Фаза полноценного статистического управления	Если количество доступных значений 30 и более, считаем, что можно определять особенные случаи, используя диапазон $\text{Mean} \pm 3\sigma$	 <p>Значение, вышедшее за пределы <math>\text{Mean} \pm 3\sigma</math>, считается особенным.</p>

Кол-во измерений	Причина (Обоснование)	Пример
Сравнение статистических параметров с заданными границами		 <p>Процесс стабилен, но его статистические параметры не удовлетворяют установленным границам.</p>

### 3. Заключение

Таким образом, количественное управление тестированием позволяет:

- на объективной основе анализировать ход выполнения процесса,
- понимать задолго до окончания процесса возможность достижения установленных целей,
- планировать в случае необходимости действия для обеспечения достижения целей,
- оценивать эффективность предпринятых мер.



# Software Estimation Tools Investigation

**Stanislav Kalkanov**  
**Luxoft**  
**Moscow, Russia**  
**SKalkanov@luxoft.com**

**Artem Babamuratov**  
**Luxoft**  
**Moscow, Russia**  
**ABabamuratov@luxoft.com**

## Abstract

Software project estimation is one of the most challenging and important area in software development. Many companies, especially those which have core-business related to software engineering, feel a need for repeatable, clearly defined and well understood estimation process. Is it so? Definitely yes. And though, from the fundamental point of view there are a lot of different formal and informal estimation technics ( from «look up to the ceiling, scratch your chin and say "well..."» to things like IFPUG and COCOMO) – many specialists doesn't know if there any good complex estimation tools on the market? What kind of estimation processes does they contain? Does it make sense to apply such solutions within organization in order to archive better predictability and rise estimates precision?

Below we've tried to consider all this questions in details.

**Keywords:** Software Estimation; Automation Tools; Comparative Analysis.

# Исследование инструментов автоматизации процесса оценки программного обеспечения

Станислав Калканов  
Люксофт  
Москва, Россия  
SKalkanov@luxoft.com

Артём Бабамуратов  
Люксофт  
Москва, Россия  
ABabamuratov@luxoft.com

## Тезисы

Оценка проектов по разработке программного обеспечения – трудная и важная задача. В настоящее время многие компании, особенно те, чей бизнес связан с разработкой ПО испытывают потребность в наличии зрелого и прозрачного процесса оценки на уровне организации.. Так ли это? Определенно – да.. И хотя,сейчас существует большое количество различных методик оценки (начиная от «это мое сугубо личное мнение» и до хорошо формализованных типа IFGUP и COCOMO) – постоянно возникают связанные с наличием комплексных решений автоматизирующих процесс оценки. Существуют-ли такие инструменты?Какой именно процесс оценки они предлагают? Имеет ли смысл их внедрять для повышения точности выполняемых оценок?

Мы постарались ответить на все эти вопросы.

**Keywords:** Оценка проекта; инструмента автоматизации; сравнительный анализ.

## 1. Введение

В сфере разработки программного обеспечения точность оценки – важный фактор, от которого зависит успешность получения качественного продукта в сроки и в соответствии с планируемым бюджетом. Построение зрелого процесса оценки программного обеспечения - одна из ключевых задач компании в мире software engineering и решать её можно несколькими путями:

- 1) Самостоятельно - обычно в основу процесса оценки закладывают различные модификации методов экспертной оценки (оценка по аналогии, wide-band Delphi и др.) и одну или несколько хорошо-известных формальных методик (Function Points Analysis, UseCase Points и др.). Для пересчета предполагаемого размера программного продукта в трудозатраты используют исторические данные или различные мат. модели, такие как Cost Constructive Model и др. Возникающие вопросы автоматизации решаются использованием доступного инструментария. Путь затратный и долгий, требует наличия глубокой экспертизы и знаний различных методов оценки.
- 2) При помощи готовых инструментов автоматизации – разработки сторонних компании, предлагающих свой процесс оценки.

Какой путь выбрать? Именно для этого в докладе будут рассмотрены результаты сравнительного анализа средств автоматизации процесса оценки и сделаны выводы об эффективности их использования.

## **2. Рассматриваемые инструменты и критерии их сравнения**

Для анализа отобран ряд как коммерческих, так и бесплатных инструментов, пользующихся наибольшей популярностью. Список включает: SLIM-Estimate, ConstruxEstimate, CoStar, CostXpert, KnowledgePlan, ObjectMerix, SEER.

Критерии, выделенные для понимания эффективности использования инструмента в Компании, занимающейся промышленной разработкой ПО:

Стоимость;

Возможность калибровки, используя историческую информацию;

Параметры, принимаемые на входе (SLOC'и, функциональные точки...);

Поддерживаемые модели расчета трудозатрат (SLIM, COCOMO, гибридные....);

Поддерживаемые жизненные циклы проектов;

Выход (общие трудозатраты, трудозатраты на этапы/работы проекта, длительность проекта, стоимость...).

#### **4. Автоматизируемый инструментами процесс оценки**

Каждый из исследуемых инструментов представляет собой «серый ящик», процесс работы с которым может быть разделен на следующие шаги:

- 1) Ввод первичной информации - на вход инструмента подается предполагаемый размер программного продукта. Это может быть как размер в строках кода, так и в других единицах (Functional Points, Feature Points, компонентах UI, функциях и т.д). Многие инструменты, в конечном счете, пересчитывают размер в кол-во строк кода. Точность оценки на входе, в большинстве своем, определяет точность оценки на выходе.
- 2) Определение общих трудозатрат - кол-во строк кода пересчитывается в общие проектные трудозатраты, вычисляется время требуемое на разработку продукта. Для этого используются уравнения моделей COCOMO II, SLIM, SEER-SEM и гибридные. Для всех уравнений задаются поправочные (калибровочные) коэффициенты.
- 3) Разбивка общих трудозатрат на конкретные проектные работы - общие проектные трудозатраты делятся на конкретные работы. В основном используются

исторические данные. В некоторых инструментах, перед выполнением оценки, можно настроить жизненный цикл проекта, обозначить основные проектные активности и их веса. Тогда на выходе получится структура работ «Work Breakdown Structure». Иногда её можно экспортировать в формат MS Project. Кроме этого можно создать большое количество отчетов.

- 4) Расчет стоимости проекта - определяется исходя из стоимости человека-часа для основных проектных ролей (задается в настройках).

Учитывая эти данные - можно говорить, что предлагаемый инструментами процесс не содержит никаких «нестандартных» или новых подходов. Для получения оценки везде используется концепция «Размер продукта->Трудозатраты».

### 3. Заключение

По совокупности итоговых оценок лидером среди коммерческих инструментов выступает продукт **CostXpert**. Это наиболее продуманная с точки зрения UI система, построенная на уравнениях модели COCOMO II. Продукт обладает гибкостью, поддерживает настраиваемые жизненные циклы проектов и позволяет полностью сопровождать процесс оценки от этапа определения размера продукта, до этапа создания плана графика проекта. Стоимость лицензии на одного пользователя **CostXpert** составляет 2970 eur / год.

В качестве бесплатного решения лучшим оказался продукт **ConstruxEstimate**, но он обладает серией серьезных недостатков: оценочная модель непрозрачна, жизненный цикл проекта предопределен и не предполагает изменений.

Важно отметить, что построение зрелого процесса оценки на уровне организации, используя даже “самую лучшую” out-of-

the-box автоматизированную систему невозможно.. И причин здесь несколько:

- 1) Для эффективной работы с рассмотренными инструментами необходимо глубокое понимание контекстной оценочной модели и практический опыт её использования. Поэтому даже самая дорогостоящая система в руках неквалифицированного оценщика, не понимающего, как она получает результат, скорее зло, позволяющее скрывать непонимание скоупа системы и манипулировать цифрами, прикрываясь брэндом автоматизированной системы. А если знания методик оценивания у Вас есть – то Вы вполне можете экономить на дорогостоящих инструментах. Тем более, что самая затратная часть процесса оценки - анализ требований. А эта работа автоматизированным средствам оценки проектов пока “не по зубам”. Поэтому здесь автоматизация не дает большого выигрыша.
- 2) Все рассмотренные инструменты являются «stand-alone» приложениями, не позволяя эффективно накапливать историческую информацию и выполнять анализ оценок с реальными трудозатратами и сроками по выполненным проектам на уровне организации. Поэтому на их основе сложно построить комплексную среду для оценки проектов.

Применение же таких систем в качестве систем поддержки выполнения оценки вместе с обязательным обучением методикам оценки, хранением и анализом исторической информации на уровне организации – даст эффективный результат.

# **Адаптивное управление компонентными программными комплексами на основе метода анализа прецедентов**

**Николай Ткачук**  
**НТУ “ХПИ”**  
**Харьков, Украина**  
**email:**  
**tka@kpi.kharkov.ua**

**Сергей**  
**Полковников**  
**НТУ “ХПИ”**  
**Харьков,**  
**Украина**  
**ausergiy@list.ru**

**Михаил Годлевский**  
**НТУ “ХПИ”**  
**Харьков, Украина**  
**god\_asu@kpi.kharkov.ua**

## **Abstract**

The architecture of adaptive software system is proposed, which is based on multi-threads programming mechanism. It includes two main logical components: the database for selected cases gathering (DBC), and the decision-making block (DMB). Using the CBR-methods the control procedure for DMB is elaborated, which utilizes the DBC and provides the appropriate algorithm defines a number of necessary threads to be implemented in order to guarantee the data processing mode with predicted performance (measured in ms), and reliability (defined in %).

**Key words:** software component, adaptive system, case-based reasoning (CBR).

## **1. Актуальность решения задач эффективного управления компонентными программными комплексами (КПК)**

Анализ современных источников по проблемам разработки программных систем (ПС) показывает, что в последнее время наметилась тенденция перехода от рассмотрения проблем реинжиниринга унаследованных систем к проблематике разработки таких подходов к проектированию и реализации сложных, т.е. распределенных и многоуровневых ПС, которые бы обеспечивали их масштабируемость, т.е. способность системы сохранять в заданных пределах ее рабочие характеристики (производительность, надежность и

т.п.) в условиях изменения вычислительной нагрузки на ее отдельные компоненты и подсистемы.

Термин “жизнеспособное программное обеспечение” (viable software) появился в серии работ австралийского ученого Ч. Херринга (Ch. Herring) [1,2]. Характерным является то, что как отмечал в одной из своих работ сам Херринг, он опирался при этом на более общую концепцию, а именно на так называемую “модель жизнеспособной системы” (viable system model), предложенную в работе С. Бира (S. Beer) и опубликованную в журнале научного сообщества по проблемам исследования операций Journal of the Operational Research Society [3].

Эти и многие другие публикации положили начало этапу активного внедрения в концептуальный инструментарий современной программной инженерии интердисциплинарных подходов, и в частности, методов теории управления и кибернетических моделей для анализа и синтеза сложных ПС. Прямым отражением этой тенденции стало появление даже такого специального термина как “программная кибернетика” (software cybernetics) (см. напр., в [4-6]). Под ним понимается подход к разработке ПО, в котором применяются такие базовые концепции вышеназванных дисциплин как использование различных схем управления с обратной связью, разработка и применение для управления компонентами ПО различных моделей знаний о предметной области, использование количественных характеристик и соответствующих метрик для оценки качества функционирования программных комплексов. Осознание эффективности применения кибернетических подходов при создании сложного ПО, в частности, принципов адаптивного управления (см., напр. в [7]), естественным образом приводит к появления адаптивных технологий в процессах разработки новых и управления уже существующими программными системами [7-10]

## **2. Классификация адаптивных технологий в современной программной инженерии**

Проблема классификации адаптивных технологий ПО представляется весьма сложной и неоднозначной. Вместе с тем, если рассмотреть ее с позиций соотнесения задач адаптации



разрабатываемой программной системы и основных фаз ее жизненного цикла (напр., в соответствии со стандартом ISO / IEC90003 который опубликован на официальном ресурсе международного консорциума по программной инженерии SWEBOK [11]), то можно мотивированно утверждать, что существуют следующие типы задач адаптации ПО, а именно: 1) адаптивность организации процессов разработки ПО; 2) адаптивность архитектуры разрабатываемой ПС; 3) адаптивность механизмов управления функционированием уже существующей ПС. Каждый из этих типов задач разработки адаптивных решений в программной инженерии, в свою очередь, может быть разделен на подтипы в соответствии с одним из трех возможных подходов к их решению, которые являются хорошо известными из общей теории адаптивных систем (см., например, в [7]), а именно: а) параметрическая адаптация; б) алгоритмическая адаптация; в) структурная адаптация. что показано на рис. 1.



Рис. 1. Классификация адаптивных технологий

С технологической точки зрения, основу для всех этих подходов составляет некоторый промежуточный слой ПО (middleware), используемый для организации гибкого взаимодействия между отдельными компонентами системы. Это позволяет говорить о так называемой композиционной адаптации (compositional adaptation) [12], которая характерна для разработки компонентных программных комплексов (КПК) с использованием таких технологий как MS COM / DCOM [13], MS .NET Framework [14], Enterprise Java Beans [15] и Web Services [16].

### **3. Механизм адаптивного управления КПК в реальном масштабе времени**

Современные КПК представляют собой сложные распределенные программные структуры, с большим числом многомерных и слабоформализуемых параметров и факторов влияния на их функционирование, например: параллельные и многопоточные процессы, колебания вычислительной нагрузки, изменения конфигурации аппаратных средств системы, числа и интенсивности работы различных групп пользователей и т.д. Все эти обстоятельства весьма затрудняют попытки построить и исследовать их точные математические модели. Говоря в терминах теории управления, не представляется возможным получить передаточную функцию КПК в аналитическом виде.

Поэтому для разработки механизмов принятия решений для адаптивного управления в КПК целесообразно применять так называемые методы «мягких вычислений» (soft calculations), к которым относятся нейросетевые технологии (neuronet technologies), методы нечеткой логики (fuzzy logic methods), генетические алгоритмы (genetic algorithms), методы логического вывода на основе прецедентов (case-based reasoning - CBR) и некоторые другие. Именно использование CBR-методов признается рядом авторов одним из наиболее эффективных подходов для разработки механизмов принятия решений при управлении сложными программными системами – см., например в [17-20]. Главная его идея заключается в предположении, что некоторую новую проблемную задачу всегда можно решить, используя или адаптируя решение уже

известной подобной проблемной ситуации (или прецедента - case), которая имела место в прошлом.

### 3.1. Формализация задачи управления КПК на основе методов анализа прецедентов

Будем далее рассматривать модель прецедента в процессе функционирования КПК в виде  $C = \langle \vec{p}, \vec{s} \rangle$ , (1), где  $\vec{p}$  - вектор параметров описания текущей проблемной ситуации (ТПС),  $\vec{s}$  - вектор соответствующих управляющих параметров. Описание ТПС задано как  $\vec{p} = \langle \tau, \eta \rangle$ , (2), где  $\tau$  - требуемая производительность или время обработки одного запроса в КПК (например, в мс),  $\eta$  - заданная при этом величина надежности обработки данных, т.е.  $(0 \leq \eta \leq 1)$ .

Вектор параметров адаптивного управления  $\vec{s}$  из модели представления прецедентов (1) для КПК типа сервера обмена данными можно в простейшем случае представить в виде  $\vec{s} = \langle n \rangle$ , (3) где  $n$  - количество вычислительных потоков, которые должны быть реализованы на сервере для разрешения данной ТПС. Тогда задача адаптивного управления КПК формулируется следующим образом: для некоторой ТПС, которая задается ее вектором  $\vec{p}$ , необходимо найти соответствующий вектор управляющих воздействий  $\vec{s}$ , имея в наличии некоторое множество прецедентов  $\mathcal{C}$ .

### 3.2. Алгоритм адаптивного управления на основе метода ближайшего соседа

Этот метод (nearest neighbor method) заключается в выборе из базы данных прецедентов (БДП) единственного прецедента, который является наиболее адекватным ТПС с точки зрения степени близости соответствующих векторов описания их параметров  $\vec{p}$ . В многомерном случае пространства описания параметров прецедентов, его применение означает выполнение следующих шагов:

Шаг 1. Предполагается, что БДП имеется некоторое количество прецедентов  $\{c_1, c_2, \dots, c_j\}$ , у которых все параметры описания проблемных ситуаций, т.е. параметры вектора  $p_j \in \mathbb{R}^L$ , являются нормированными по формуле

$$\bar{x}_{ij} = \frac{x_{ij} - \min_j \mathbf{C}_{ij}}{\max_j \mathbf{C}_{ij} - \min_j \mathbf{C}_{ij}}, \quad i \in \llbracket L \rrbracket, j \in \llbracket P \rrbracket, \quad (4)$$

где  $\bar{x}_{ij}$  - значение  $i$ -ого параметра для  $j$ -ого прецедента,  $L$  - мощность множества параметров описания вектора  $p_j \in \mathbb{R}^L$ , т.е. вектора параметров описания проблемной ситуации для каждого прецедента,  $\max_j \mathbf{C}_{ij}$  и  $\min_j \mathbf{C}_{ij}$  - соответственно,

максимальное и минимальное значение этого параметра среди всех прецедентов, присутствующих в БДП, а  $P$  - мощность их множества.

Шаг 2. На вход алгоритма подается описание ТПС, которая возникает в операционной среде функционирования КПК в момент времени  $t$ , в виде совокупности нормированных значений их параметров  $\mathbf{x}^{(t)}$ , которые в свою очередь определяются аналогично выражению (4).

Шаг 3. Определяется мера близости ТПС относительно каждого из имеющихся в БДП прецедентов по всем его признакам, для чего, при некоторых упрощающих допущениях, можно использовать метрику в виде Евклидова расстояния

$$d_j^{\mathbf{C}} = \left( \sum_{i=1}^L (\mathbf{C}_{ij} - x_i^{\mathbf{C}})^2 \right)^{\frac{1}{2}}, \quad i \in \llbracket L \rrbracket, j \in \llbracket P \rrbracket, \quad (5)$$

Шаг 4. В БДП выбирается прецедент с минимальным расстоянием до ТПС, а именно, такой прецедент, для которого

$$d_0^{\mathbf{C}} = \min_j \mathbf{d}_j^{(t)}, \quad j \in \llbracket P \rrbracket \quad (6)$$

Шаг 5. В качестве искомого решения принимается вектор управляющих параметров  $\vec{s}_k$ , соответствующий найденному

прецеденту, находящемуся на расстоянии  $d_0^{(t)}$  от рассматриваемой ТПС.

### 3.3. Архитектура адаптивного КПК

Для программной реализации адаптивного механизма управления КПК предлагается архитектура, представленная на рис. 2. В ней предусмотрено наличие трех параллельных вычислительных потоков, а именно: 1) поток формирования запросов к системному буферу данных (СБ), который может выполнять это действие в синхронном либо в асинхронном режимах; 2) поток обработки полученных ответов (блоков данных); 3) поток генерации управляющих решений, который непосредственно обеспечивает возможность функционирования блока адаптивного управления (БАУ). В свою очередь, в составе БАУ выделяются следующие функциональные блоки: блок формирования параметров описания текущей проблемной ситуации (ТПС), которая возникает в процессе функционирования данного КПК; база данных прецедентов (БДП), которая хранит в себе описания ранее накопленных прецедентов; блок поиска решений (БПР), который используя методы логического вывода на основе анализа прецедентов производит поиск подходящего прецедента в БДП.

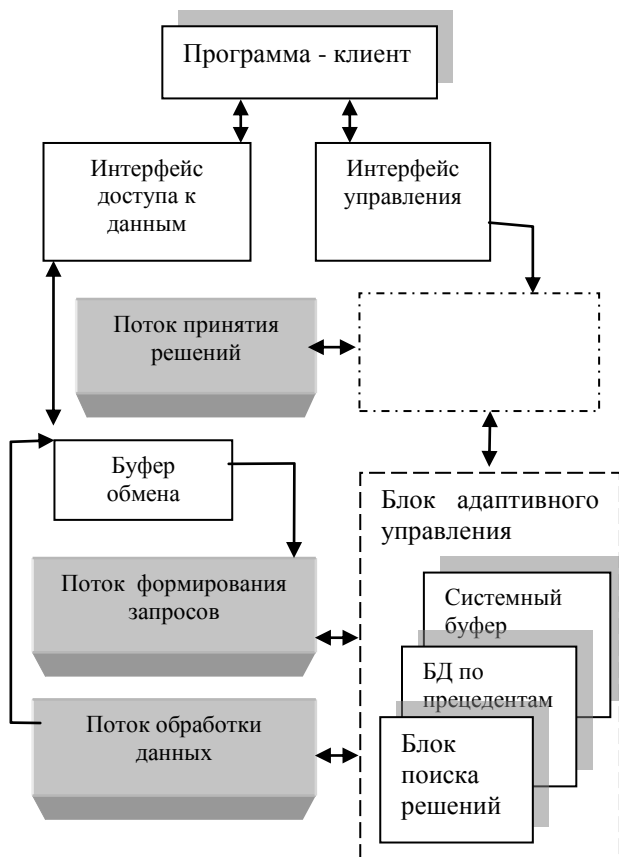


Рис. 2 Архитектура адаптивного КИПК

#### 4. Экспериментальная реализация и исследование эффективности предложенного решения

Бизнес-логика работы БАУ представлена диаграммой вариантов использования на рис. 3.

На ней показаны все основные возможности, которые предоставляет разработанное программное решение, а именно: создание прецедента, извлечение всех прецедентов из БДП, извлечение прецедентов по некоторому критерию поиска и определение наиболее подходящего значения параметра адаптируемого КПК на основе прецедентов, имевших место в системе. Для программной реализации этого решения была выбрана платформа Java (с целью эффективного решения проблемы многопоточности) с использованием таких open source-проектов как Spring и Hibernate, а в качестве СУБД для работы с БДП – открытая система MySQL.

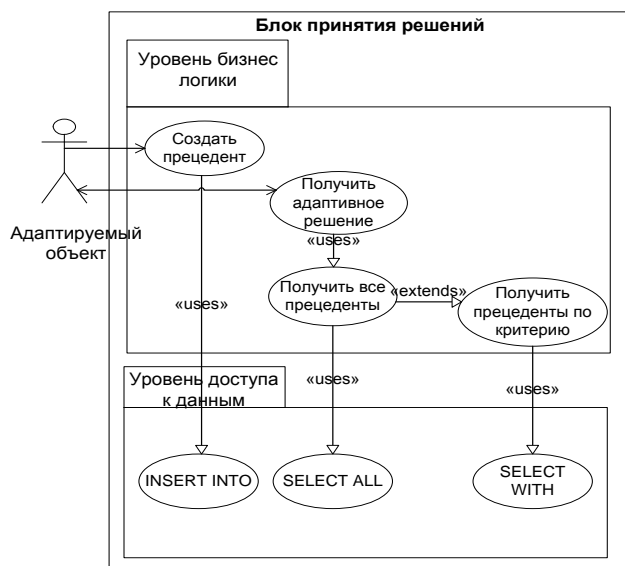


Рис. 3. Бизнес-логика работы БАУ

Для проверки работоспособности предложенной процедуры нахождения адаптивного управляющего решения были использованы экспериментальные данные, полученные с помощью специального программного имитатора. Они представлены в таблице 1.



Таблица 1 (фрагмент)  
Данные для проверки работоспособности БАУ

Кол-во потоков (n)	Время отклика (мс)	Надежность [0,1]
1	85.41	1
2	55.62	1
3	31.84	0.99
4	29.26	0.99
5	22.28	0.98
6	20.32	0.98
7	19.44	0.97
8	17.48	0.97
9	15.87	0.96
10	15.63	0.96
.....	.....	.....

В качестве параметров описания ТПС были рассмотрены следующие значения: требуемая производительность сервера, равная времени обработки одного запроса  $\tau = 30$  мс; надежность обработки данных  $\eta = 0.98$ , которые должны быть обеспечены для его адекватного функционирования в условиях данной ТПС.

В табл. 2 показаны результаты расчетов по алгоритму поиска требуемого решения, представленному в п. 3.2 (см. формулы (4) – (6)).

Таблица 2 (фрагмент)  
Результаты расчета по CBR - методу

Кол-во потоков (n)	Мера близости к ТПС	Вес прецедента
1	0.64902018	2.3741
2	0.30043223	11.0793
3	0.02332921	1837.3922
4	0.00936767	11395.6112
5	0.09033121	122.5538
6	0.11402252	76.9144
7	0.12400521	65.0315
8	0.14796215	45.6771
9	0.16662936	36.0169
10	0.17126622	34.0935
.....	.....	.....

Анализ этих данных показывает, что при использовании метода в качестве решения находится прецедент с количеством потоков равным  $n = 4$ , т.к. он имеет наилучшую меру близости и наибольший весовой коэффициент (см. табл. 2). Наглядно этот результат показан на соответствующем графике – см. рис. 4.



Рис. 4. Результаты применения CBR-метода

Точность поиска адаптивного решения, определяемая выражениями (4)-(6) может быть улучшена путем применения метода  $k$ -ближайших соседей (см., напр. в [21]).

## 5. Выводы и направления дальнейших исследований

Таким образом, в результате экспериментальной проверки разработанного подхода показана его работоспособность для решения задачи нахождения адаптивного решения по организации многопоточной работы КПК типа сервера обмена данными, которое обеспечивает выполнение заданных значений параметров его функционирования для некоторой текущей проблемной ситуации.

Следует отметить, что в данной постановке решения задачи нахождения адаптивного управления для функционирования распределенной ПС не учитывались такие весьма важные факторы, существенно влияющие на ее производительность и надежность, как, например:

- возможность одновременного использования в системе нескольких серверов приложений;
- применение различных механизмов кэширования на каждом из таких серверов соответствующих запросов клиентских приложений;
- учет фактора влияния пропускной способности сетевых соединений и некоторые другие.

Моделирование и экспериментальное исследование этих задач в контексте решения общей проблемы разработки эффективных механизмов для адаптивного управления распределенными ПС и представляют собой направления дальнейших работ авторов данной статьи.

## **6. Список использованных источников**

- [1] Ch. Herring, S. Kaplan, “The Viable System Architecture” // Proceedings of Thirty-Fourth Hawaii International Conference on System Sciences (HICSS-34), 2000. Maui, Hawaii.
- [2] Ch. Herring, S. Kaplan, “The Viable System Model for Software” // Proceedings of 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000), 2000. Orlando, Florida.
- [3] S. Beer, “The Viable System Model: Its Provenance, Development, Methodology and Pathology”, Journal of the Operational Research Society, 1984.
- [4] S.D. Miller, R.A. DeCario, A.P. Mathur, “A Software Cybernetic Approach to Control of Software System Test Phase” // Proceedings of the 29<sup>th</sup> Annual International Computer Software and Applications Conference, 2005.

- [5] M. Aksit, Z. Choukair, "Dynamic, Adaptive and Reconfigurable Systems Overview and Prospective Vision" // Proc. 23<sup>rd</sup> International Conference IEEE CS Press, May 2003.
- [6] McKinley P.K. et al. A Taxonomy of Compositional Adaptation // Techn. Report MSU-CSE-04-17. Dept. CS and Engineering, Michigan State Univ., 2004.
- [7] Поспелов Д.А. Логико-лингвистические модели в системах управления. – М.: Энергоиздат, 1981.
- [8] О.В. Савченко, А.В. Трубецкой, В.В. Неверов, "Концепция построения системы адаптации вычислительного процесса в многоцелевых АСУ", Информация и космос, №1, 2005.
- [9] Л. Черняк, "Адаптируемость и адаптивность", Открытые системы, №9, 2004.
- [10] Ткачук Н.В. Модели, методы и информационные технологии адаптивной разработки и реинжиниринга информационно-управляющих систем // Автореф. дисс. на соис. ученой степени д-ра техн. наук., – НТУ «ХПИ», Харьков, 2005.
- [11] Интернет-ресурс международного консорциума по программной инженерии SWEBOK [www.swebok.org](http://www.swebok.org).
- [12] Ф. Маккинли, Э. Кастен, Б. Ченг, "Композиционная адаптация программ", Открытые системы, №9, 2004.
- [13] Бокс, Д. Сущность технологии СОМ. - СПб: Питер, М.: «Русская редакция», 2001.
- [14] Просиз Дж. Программирование для Microsoft.NET . – М.:Русская редакция, 2003.
- [15] Ален Р., Бамбара Дж., Ашнаульт М. и др. J2EE: Разработка бизнес-приложений - СПб. :ДиаСофтЮП, 2002.

- [16] Феррара ,А. Мак-Дональд М. Программирование Web-сервисов для .NET - СПб. :Питер, 2003.
- [17] Юдин В. Н. Исследование и разработка системы поддержки принятия решений на основе прецедентов // Автореф. дисс. канд. техн. наук: 05.13.11, Ин-т системного программирования РАН — М.:, 2007.
- [18] Л. Е. Карпов, В. Н. Юдин, ”Адаптивное управление по прецедентам, основанное на классификации состояний управляемых объектов”, Препринт Института системного программирования РАН, - 2007
- [19] B. Limthanmaphon, Y. Zhang, ”Web Service Composition with Case-Based Reasoning” / Fourteenth Australian Database Conference (ADC2003), Adelaide, Australia, Conferences in Research and Practice in Information Technology, Vol. 17. - Australia. – 2004.
- [20] А.И. Павлов, А.Ю. Юрин. Программный модуль правдоподобного вывода по прецедентам - <http://www.ict.nsc.ru/ws/YM2007/12681/Yurin.htm> - Препринт Института вычислительных технологий Сибирского отделения РАН.
- [21] Аль-Хассание Захер. Модели и информационные технологии для адаптивного управления компонентными программными комплексами // Автореф. дисс. на соис. ученой степени канд. техн. наук., – НТУ «ХПИ», Харьков, 2008.

# Application of Rewriting Term System for Source Code Analysis

**Ruslan Shevchenko**  
**GradSoft Ltd**

**ruslan@shevchenko.kiev.ua**

**Anatoliy Doroshenko**  
**INTSPEI**

**doroshenko@intspei.com**

## **Abstract**

A extensible source code manipulation framework on top TermWare rule system is introduced and its application to software system analysis is presented. The framework allows to build light-weighted formal models, focused on particular properties of a program to be analyzed and does not need full computational semantics of the program. This allows building a number of useful practical approximations of computational semantics in inexpensive and efficient way. Two practical applications – static analyzer for Java source code and a system for partial evaluation of Java sources are described.

**Keywords:** TermWare, rule-based systems, source transformations, static analysis

## **1. Introduction**

Rewriting rule based techniques are well known to be used in symbolic computation as well as in software system development. A particular class of rewriting rule systems consists of comprehensive ones such as Maude[1], Stratego[2] or APS[3] that have appeared as an alternative to logic programming systems, Jess [4] intended to use rewriting rules for construction of expert system, and ASF+SDF [5] for transformations of syntactical trees analysis accordingly. The system Tom [6] is extension of Java to manipulate tree structures.

A reference feature of rewriting rule systems is availability of the rules to represent subject domain objects as formal expressions (e.g. algebraic terms) and universal capability of transforming these terms

through the policies of applying production rules based on non-logic principles, often in a form of imperative sequences (or strategies) of rule applications.

TermWare system has been recently developed and applied by the authors in different subject domains of software engineering [7], [8], [9]. It also belongs to comprehensive rewriting systems but differs from other systems of this class in both semantics of used facilities and technology of their implementation.

TermWare language is not a universal programming language for writing full-fledged software system. Instead it is aimed to compose domain specific parts of the applications built into an applied system for implementing functions of interaction of the system with its environment.

The semantic features of TermWare language also essentially differs from conventional rewriting systems: Usually rewriting rules programmer uses the formal model of some subject domain to implement classic “closed” computation paradigm in functional style where input/output operations and interactions are not represented explicitly and are rather “side effects” than regular events.

In TermWare formalism includes not only set of rewriting rules, but interactions with environment that is represented as dynamic base of the facts. Thus, instead of hiding imperative style of programming in side effects TermWare offers some kind of immersion of imperative operations into logic of declarative program. The term system is open in the sense that it is interactive and its behavior is determined not only by input terms, but by state of environment.

The outline of the paper is as follows. In the next section the formal model of TermWare language and the system are presented. An example of TermWare application in semantic approximation while analyzing source code for defect detection is described in section 4. Concluding remarks are given in section 5.

## **2. TermWare system**



The model of TermWare is constructed as algebra of terms by expressions of the form  $f(x_1 \dots, x_n)$  with variables and basic data types similarly to other typeless functional programming languages. Some well known functional names can be shortcut with infix syntax ( $x+y$  instead  $\text{plus}(x,y)$ )

Computation is defined by sets of the rewriting rules together with policy of their application. Unlike traditional rewriting systems, rewriting rule is not a pair  $x \rightarrow y$ , but a four  $x \text{ [ in ]} \rightarrow y \text{ [out]}$ , where  $x$ ,  $y$  - input and output terms; in, out - input and output reactions. Semantics of rule can be interpreted as imperative describer "rewrite  $x$  to  $y$  under condition in having applied to environment operation out". Environment is defined as class in object oriented language (Java in our case), so in and out are terms which includes calls for methods of environment class.

The main entity of termware language is the term system, which consists from of set of rules, environment, name and strategy. I.e. the term system itself is defined as a term of the form

$\text{System}(\text{name}, \text{ruleset}, \text{facts}, \text{strategy})$

where name is a name of the system, ruleset is a set of rewriting rules with interaction, facts is a database of the facts of environment, strategy is an algorithm of ruleset application. Thus, TermWare gives a direct mapping of logical model of interactions to the programming language having clones of modern means of support of inheriting and a hierarchical name system.

Detailed description of TermWare semantics is presented in [7].

### **3. Representation of source code.**

#### **3.1. AST terms.**

The first naive, but practical approach to source code analysis presents an abstract syntax tree (AST) as a term in TermWare system. This term can be manipulated by means of facilities provided by TermWare system. Such representation can be seen as some form of symbiosis of AST and algebraic terms with opportunities of performing well-defined operations expressed as declarative rewriting rules.

As an illustrative example consider the following Java code:

```
package x;

class X
{
    int x() { return 1; }
}
```

which corresponds to the following term:

```
CompilationUnit(
  PackageDeclaration(Name(cons(Identifier("x"),NIL))),
  TypeDeclaration(Modifiers(1,NIL),
    ClassOrInterfaceDeclaration(
      class
    ,Identifier("X")
    ,NIL,
      ExtendsList(NIL),
      ImplementsList(NIL),
      ClassOrInterfaceBody(
        cons(
          ClassOrInterfaceBodyDeclaration(
            Modifiers(0),
            MethodDeclaration(NIL,
              int
              MethodDeclarator(
                Identifier("x"),
                FormalParameters(NIL))
            )
          )
        )
      )
    )
  )
)
```

```

,NIL,

Block(
  cons(
ReturnStatement(
  IntegerLiteral("1")),
  NIL)))

),

NIL

))))))

```

To walk over such terms one can use rewriting rules. For example following rule from JavaChecker project [8] triggers warning message on empty catch blocks:

```
catch($x,Block(NIL)) -> true
```

```

[ violationDiscovered(EmptyCatchBlock,
  "empty catch block",$x) ].

```

### 3.2. Model terms

As one can see from previous examples syntax only manipulations are quite limited: Computations are local and one can check only relations with terms which are situated 'near' each other. Symbol categorization is absent; for example, in the sentence `this.x=x` we can not distinct member variable in the left part from parameter (or local variable) in the right part of the sentence.

TermWare solution for this problem is providing 'right' form of source representation, which include not only syntax tree, but some global context access object(`mark`) which involve functionality of semantics categorization of symbols and navigation on relations between them in semantics scope of analyzed program.

Note, that exists another approach implemented in stratego system[10]: so called dynamic rules, which can be created and removed in runtime in depends of scope of processing. Such approach can solve problem of context availability (but not navigation). So, TermWare provide model representation, differs from AST by embedding some extra parameter – source context which provides metainformation about current expression (similar to Java modeling elements of API in JSR 269 [11]).

Since TermWare defines mapping of Java classes into terms, we can use object API for access to such information still preserving declarative style of rules.

For this purpose for each term  $x$  we will define the model term  $\text{model}(x)$  in following way:

- if  $x=f(x_1, \dots, x_n)$  then  $\text{model}(x)= f\_m(\text{model}(x_1), \dots, \text{model}(x_n), \text{ctx})$ , where  $f\_m$  is appropriate model symbol and  $\text{ctx}$  is a special object which provides access resolver and context of current expression.
- if  $x$  is literal then  $\text{model}(x)=x$ .

For example, the model term for `X.java`, mentioned in previous section looks like following:

```
ClassOrInterfaceModel(
  Modifiers(1),class,Identifier("X"),
  NIL,NIL,NIL,
  ClassOrInterfaceBody(
    cons(MethodModel(Modifiers(0), NIL,
      TypeRef(int,
        jobject(JavaPrimitiveTypeModel@179dce4)
      ),
      Identifier("x"),
```

```

FormalParameters(NIL),
NIL,
cons(
    ReturnStatementModel(
        IntegerLiteral("1"),
        jobject(JavaPlaceContext@19bb25a)),
        NIL),
        jobject(@1e58cb8)),
    NIL)),
    jobject(@179935d)
)

```

where `JavaPlaceContext` encapsulates API for resolving of symbol terms to access semantics model of analyzed program. In this way our environment is rich enough to build strict algorithms of abstract interpretation, up to partial evaluation.

## 4. Examples of usage

### 4.1. Detection of resource leaks

As an example, let us look on semantic approximation for resource leaks detection. Resource leak is a phenomenon of requested resource from operation system and middleware that is not released thereafter. Let have a look at following sample code:

```

public static void main(String[] args) {

```

```

FileReader reader = null;

try {

    reader = new FileReader(args[1]);

} catch(FileNotFoundException ex){

    ex.printStackTrace();

    return;

}

for(int i=0; i<10; ++i) {

    int ch=0;

    try {

        ch=reader.read();

    } catch(IOException ex){

        ex.printStackTrace();

        return;

    }

    System.out.print(ch);

}

if (reader!=null) {

    try {

        reader.close();

    } catch(IOException ex){

```

```

        ex.printStackTrace();

        return;

    }

    System.out.print(ch);

}

if (reader!=null) {

    try {

        reader.close();

    }catch(IOException ex){

    }

}

}

```

Here if an exception is occurred inside for loop than FileReader reader remains unclosed. For the purpose of leak determination we can abstract program state as sequence of variables, which are opened, but not closed in current scope. Then, for each language construction we can either generate a set of possible states or consider it as an operation over a state.

Let the state will be one of:

1. OPENED(expr, fileAndLine) which means that expr is in opened but not closed (where expr is an expression model, usually local variable)
2. cons( $s_1, s_2$ ), where  $s_1, s_2$  are states. This means that  $s_1$  is applied after  $s_2$ .

3. RETURN(s), which means that execution flow is ended with the state s.
4. cond(v,s), which means that the state s is possible when v is true.
5.  $s_1 || s_2$ , which means one of states  $s_1$  or  $s_2$ . (But we does not known which one)
6.  $s_1 \&\& s_2$ , which means, that we must check each from states  $s_1$  and  $s_2$

Final state is correct (i. e. program is executed without resource leaks) if the resulting state does not contains OPENED elements.

Now, let's map Java model term to the execution semantics. A class is considered correct if all initializers and methods of this class are correct. Block is just a sequence of statements, and the real work begins at the level of statements and expressions.

Among all expressions, the most important for resource leak detection are allocation expressions and method calls. We think, that allocation of variable cause OPENED state, when we create object, which implements Java Closeable interface. Let's assume, that we create a new Closeable object on method call, if method can be named as so called 'Factory Method'. With allocation expressions situation is more interesting. In general, not every unclosed instance of Closeable interface can cause resource leak: there may exist some 'fake' Closeables which must implement interface but does not use system resources, such as StringWriter. Yet one problem is so-called 'ProxyAllocation' when the code from first fragment cause resource leak, but the code from the second fragment – not.

Fragment 1:

```
PrintStream x =
    new PrintStream(new FileStream(fname));
```

Fragment 2:



```
StringOutputStream ss =
    new StringOutputStream();

PrintStream x = new PrintStream(ss);
```

So, for resolving such problems the program model must contain information not only from source code, but from some other information sources which can not be deduced from code. For this purpose mechanism of external annotations has been created which makes possible to annotate classes and methods and does not touch existing source code.

```
CheckExpression($var,
```

```
AllocationExpressionModel(TypeRef($name,$type),$arguments,$ctx),$
state)
```

```
[
    $ctx.subtypeOrSame(
        $type,

        $ctx.resolveFullClassName(
            "java.io.Closeable"))

    &&

    !($type.getAttribute(
        "NotCloseable")==true)
]
-> CheckAllocationNotProxy(
    $var,$type,$arguments,
    $ctx,$state)

!-> CheckExpressions($arguments,
    $state),
```

```

CheckAllocationNotProxy($var,$type,
                        $arguments,$ctx,
                        $state)

->

((
    !($type.getAttribute(
        "CloseableProxy")==true)

    ||

    apply(
        openclose::CloseableArguments,

        apply(NormalizeExpressions,
            $arguments))

    ))

?

cons(
    OPEN($var,$ctx.fileLine)
    , $state)

:
    CheckExpressions(
        $var,$arguments,$state),

```

This is a rule for allocation expression. At first, look at conditions: by call of subtypeOrSame we determine if allocated type is inherited from java.lang.Closeable. The attribute NotCloseable is a special attribute which must be set for classes, safe to leave without call of close(), such as java.io.StringWriter(). If such condition is met then we check that allocation is not 'ProxyAllocation' and add OPENED(x) to list of states.

As example of other kind of rule – the rule for processing if statement:

```

CheckStatement(
  IfStatementModel($expr,$s1,$s2,$ctx),
  $state) ->

  let ($evstate <- $state )

cond(
  $expr,
  CheckStatement($s1,
    CheckExpression(NONE,$expr,$evstate))

  &&

  cond(not($expr),
    CheckStatement($s2,

      CheckExpression(NONE,
        $expr,$evstate))))).

```

As one can see this rule looks like a part of interpretation framework. At some level of abstraction, tracking resources can be described as abstract interpretation over states.

Now, our abstract state can be simplified, when we calls close method. This is defined in next rule fragment:

```

CheckExpression($var,
  MethodCallExpressionModel($obj,
    $methodModel,
    $arguments,$ctx),
  $state)

.....
[

  $ctx.subtypeOrSame(

```

```

    ModelHelper.getType($obj),

    $ctx.resolveFullClassName(
        "java.io.Closeable")

)

&&

$methodModel.getName()=="close"

]

->
    merge(CHANGE($obj,CLOSE),$state)
....

```

where merge is an operator on abstract state, defined by set of rules, which perform actual simplification of program state.

```

merge(CHANGE($x,CLOSE),[]) -> [ ],

merge(CHANGE($x,CLOSE),[$y:$z])
-> [checkOpenClose($x,$y),
    merge(CHANGE($x,CLOSE),$z)]
and so on.

```

## Specialization via partial evaluation

As one more application, consider a specialization of Java programs using simplified form of partial evaluations. This is natural way of extending previously described functionality, only one new element – pretty printer for Java terms.

Specialization of partial evaluation can be described as a process of mapping some Java program A and set of constants C to Java program  $\text{mix}(A,C)$  which works exactly as A on all datasets that contains C. This

is some simplification of real partial evaluation, because we have varying set of constants instead input.

Typical rules are just recursive downhill of partial evaluation marker:

```
JPE(ClassOrInterfaceModel(
```

```
    $modifiers,
```

```
    $type,
```

```
    $name,
```

```
    $typeParameters,
```

```
    $extendsList,
```

```
    $implementsList,
```

```
    $body,
```

```
    $context))
```

```
->
```

```
ClassOrInterfaceModel(
```

```
    $modifiers,
```

```
    $type,
```

```
    $name,
```

```
    $typeParameters,
```

```
    $extendsList,
```

```
    $implementsList,
```

```
    JPE($body),
```

```

    $context),

JPE(ClassOrInterfaceBody($x))
  -> ClassOrInterfaceBody(JPE($x)),

JPE([]) -> [],

JPE(cons($x,$y))
  -> cons(JPE($x),JPE($y)),

```

When we change on expression level, JPE ruleset contains rules for performing constant computation:

```

JPE(AndExpressionModel($x,$y,$ctx))
  ->

    AndExpressionModel(JPE($x),
                        JPE($y),$ctx),

AndExpressionModel(
  BooleanLiteral(false),$y,$ctx)

  ->
    BooleanLiteral(false),

AndExpressionModel($x,
  BooleanLiteral(false),$ctx)

  -> BooleanLiteral(false),

AndExpressionModel(BooleanLiteral(true),
  $y,$ctx) -> $y,

AndExpressionModel($x,
  BooleanLiteral(true),$ctx) -> $x,

```

And we must substitute variables from C by their values:

```
JPE(FieldModel($obj,$id,$fm,$ctx))
```

```
[ isJPEField($fm,$y) ] -> $y
```

```
[isStaticFinalLiteralField($fm,$y)] -> $y
```

```
!-> FieldModel($obj,$id,$fm,$ctx),
```

Process of partial evaluation implemented in TermWare transformation rules performs simplification of model terms and their transformation to syntactic AST terms. It can also be represented as abstract interpretation, where abstract states are specialized sources. All functionality is described by near 200 rules in less than 1000 lines of code.

Practical usage of partial evaluation is not only specialization but also optimization transformations which can be deduced from the fact that program works in specialized environment. Two main optimizations are method call devirtualization and elimination of unreachable code.

Elimination of unreachable code includes reachability analysis and removing

classes and methods, unreachable from application entry point. This optimization is implemented as imperative algorithm, which runs after specialization.

Method call devirtualization is substitution of virtual method calls by non-virtual when program analysis shows that concrete type of method is statically known. In some cases such transformation can cause significant speedup of program execution. One of simple devirtualization strategies is just marking as final all classes which have no childs. Availability of childs can be checked by global source analysis. Note, that such optimization can not be performed by compiler or virtual machine, because compiler does not perform global analysis.

## 5. Conclusion

In this paper the term-rewriting framework TermWare and its application to a real non-trivial problem in source code analysis and

transformations are considered that are used in industrial strength projects [8],[9]. The combination of imperative and declarative approach appears to be more powerful than declarative-only or imperative-only solution.

We believe that multiparadigm programming is the way to actually bring rewriting techniques to programming mainstream. Our future research efforts will be focused on better integration of object orientation, more precise model of Java semantics (including stack emulation) and working with concerted models of multiple target languages (especial SQL and C#) to obtain multi-language transformation framework.

## References

- [1] Timothy Winkler, "Programming in OBJ and Maude", in Functional Programming, Concurrency, Simulation and Automated Reasoning, International Lecture Series 1991--1992, Springer-Verlag, McMaster University, Hamilton, Ontario, Canada, 1993, pp. 229-277
- [2] Visser E, "'Stratego: A language for program transformation based on rewriting strategies. System description of Stratego 0.5", Rewriting Techniques and Applications (RTA 01) LNCS vol. 2051, Springer, 2001 pp. 357-361
- [3] J.V.Kapitonova A.A.Letichovsky M.S.L'vov and V.A.Volkov. "Tools for solving problems in the scope of algebraic programming", LNCS vol. 968, Springer-Verlag, 1995
- [4] Ernest Friedman-Hill. "Jess in Action", Manning Publications Co, 2003
- [5] M. van den Brand and J. Heering and P. Klint and P. Olivier. "'Compiling Language Definitions: The {ASF}+{SDF} compiler", University of Amsterdam, 1999
- [6] Balland, Emilie and Brauner, Paul and Kopetz, Radu and Moreau, Pierre-Etienne and Reilles, "Tom: Piggybacking Rewriting on Java", Proceedings of the 18th Conference on Rewriting Techniques", {Springer-Verlag, 2007
- [7] Anatoly E. Doroshenko and Ruslan Shevchenko. "A Rewriting Framework for Rule-Based Programming Dynamic Applications", Fundam. Infom vol. 72, 2006, pp. 95-108.
- [8] Ruslan Shevchenko. "JavaChecker homepage: [http://www.gradsoft.ua/products/javachecker\\_eng.html](http://www.gradsoft.ua/products/javachecker_eng.html)", GradSoft
- [9] Ruslan Shevchenko and Anatily Doroshenko. "Managing Business Logic with Symbolic Computations", Lecture Notes in Informatics V. 30



- Proceeding of Information Systems Technology and its Applications 2003, Gesellschaft fur Informatik 2003

[10] Martin Bravenboer and Arthur van Dam and Karina Olmos and Eelco Visser. "Program Transformation with Scoped Dynamic Rewrite Rules", Fundam. Inf. vol. 69 num 1-2, IOS Press, Amsterdam, The Netherlands, 2006

[11] JSR 269: Pluggable Annotation Processing API., Java Community Process, Sun Microsystems,

# Automata Classes Inheritance in Dynamic Language Ruby

**Artyom Astafurov (DataArt), Kirill Timofeev (DataArt), Anatoly  
Shalyto (SPbSUITMO)**  
**artyom.astafurov@gmail.com**

## Abstract

Automata-based programming is often used for creating systems with the complex behavior. However the automata-based programming has problems such as support of automata-based code and documentation, further system improvements and code clarity. Object-automata approach is based on the object-oriented and the automata-based approaches and combines their main advantages, such as a flexibility, scalability and availability of a powerful mechanism for describing complex systems based on the finite state machines. This approach allows to resolve the problems described above.

In existing object-automata approaches it is hard to encapsulate the transition clauses and develop code that clearly reflects the transitions between states as the transition logic is often hidden in the handler methods of incoming actions. In order to resolve the problems of existing object-automata approaches and retain all their benefits it is suggested to use dynamic programming languages for creating automata programs. This paper describes two approaches based on object-automata approach that use object-oriented and dynamic Ruby properties.

This paper shows that using object-oriented features of Ruby each state and automata can be represented as a separate class. It helps to maintain the automata hierarchy in the object-oriented code. However this approach has disadvantages, such as syntax redundancy that complicates code modifications and improvements.

With the help of Ruby's dynamic language features a domain-specific language (DSL) can be developed. The DSL will allow to an isomorphic translation of state charts to code and solve problems of pure object-oriented approach. However this approach has a disadvantage of losing the automata hierarchy because each state and automata are not represented as a separate class.

**Keywords:** Automata-based programming; object-oriented programming; object-automata approach; functional programming; dynamic languages; domain-specific languages; DSL; Ruby.

# **Наследование автоматных классов с использованием динамических языков программирования на примере Ruby**

**Артем Астафуров(DataArt), Кирилл Тимофеев (DataArt),  
Анатолий Шалыто (СПбГУ ИТМО)  
artiom.astafurov@gmail.com**

## **Тезисы**

При создании систем со сложным поведением целесообразно применять автоматное программирование. Однако при его использовании возникают проблемы, связанные с поддержкой автоматного кода и документации, с внесением изменений в систему, а также наглядностью и понятностью автоматного кода. Объектно-автоматное программирование основано на объектной и автоматной парадигмах программирования. Оно совмещает в себе их основные преимущества: гибкость, расширяемость и наличие мощного механизма описания сложного поведения, основанного на конечных автоматах, – что позволяет решить описанные выше проблемы.

Однако в существующих объектно-автоматных подходах не всегда удается удобно выделять условия переходов и писать код, который наглядно отражает переходы между состояниями, так как логика переходов обычно скрывается в методах-обработчиках входных воздействий. Для устранения недостатков объектно-автоматного подхода, а также сохранения всех его достоинств, рассматривается применение динамических языков программирования для построения автоматных программ. В данной работе описаны два подхода, использующие объектно-ориентированные и динамические свойства языка Ruby.

Показано, что достоинством использования объектно-ориентированных свойств языка Ruby является то, что каждое состояние и автомат представлены отдельным классом. Это позволяет сохранить иерархию автомата при переносе его в объектно-ориентированный код. Однако этот подход обладает таким недостатком, как синтаксическая избыточность, что иногда может затруднять модификацию и расширение кода.

При помощи динамических свойств языка Ruby может быть разработан предметно-ориентированный язык (DSL), который позволяет изоморфно переносить диаграммы состояния в программный код, и решает проблемы предыдущего подхода. Однако недостатком динамического подхода является потеря иерархии родительского автомата при наследовании, так как состояние и автомат не представлены отдельными классами.

**Keywords:** Автоматное программирование; объектно-ориентированное программирование; объектно-ориентированное программирование с явным выделением состояний; функциональное программирование; динамические языки программирования; предметно-ориентированный язык; DSL; Ruby.

## 1. Введение

При создании систем со сложным поведением целесообразно применять автоматное программирование называемое также «программирование от состояний» или «программирование с явным выделением состояний» [1]. Однако при использовании автоматного программирования возникают проблемы, связанные с поддержкой автоматного кода и документации, с внесением изменений в систему, а также наглядностью и понятностью автоматного кода.

При совместном использовании объектной и автоматной парадигм программирования этот подход был назван в работе [1] «объектно-ориентированным программированием с явным выделением состояний», которое в дальнейшем будем называть «объектно-автоматное программирование».

Объектно-автоматное программирование совмещает в себе основные преимущества объектной и автоматной парадигм, такие как гибкость, расширяемость и наличие мощного механизма описания сложного поведения, основанного на конечных автоматах, — что позволяет решить описанные выше проблемы.

Наследование и вложение состояний, применяемое в объектно-автоматном программировании, позволяет значительно сократить требуемое число переходов для описания сложного автомата [2].

Однако в описываемых в этой статье объектно-автоматных подходах не всегда удастся удобно выделять условия переходов, а также писать код, который наглядно отражает переходы между состояниями, так как логика переходов обычно скрывается в методах-обработчиках входных воздействий.

Для устранения этих недостатков объектно-автоматного подхода, а также сохранения всех его достоинств, в данной работе предлагается рассмотреть применение динамических языков программирования для построения автоматных программ. В качестве такого языка будет использован язык Ruby.

В данной работе будут описаны два подхода к разработке автоматных программ на языке Ruby, а также выполнено их

сравнение с уже существующими решениями. В качестве примера предлагаемых подходов рассмотрено создание автоматных классов для чтения и записи в файл с использованием вложения и наследования.

## **2. Обзор подходов и решений реализации автоматов**

Существует несколько подходов к реализации автоматов: полностью ручное программирование, автоматическая генерация кода по диаграмме переходов и ручное написание кода с применением специальной библиотеки [3].

В приведённых ниже работах используется ручное написание кода с применением специальной библиотеки для реализации автоматных классов, однако каждое из указанных решений наряду с достоинствами, обладает и недостатками.

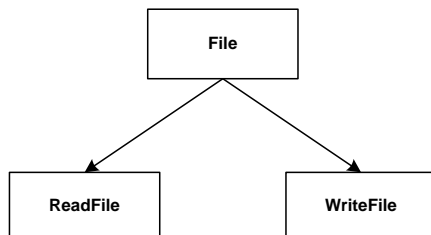
1. В работе [3] используется динамический язык программирования Ruby, с помощью которого была разработана библиотека STROBE, что позволило удобно отобразить автомат из графической нотации в программный код. Однако в этой работе не было реализовано наследование автоматных классов.

2. В работе [4] применяется декларативный подход к реализации автоматных объектов при помощи объектно-ориентированных императивных языков программирования со статический проверкой типов. Используя язык программирования C#, было реализовано наследование и вложение автоматов. В то же время код программы обладает синтаксической избыточностью, что может затруднять модификацию логики переходов.

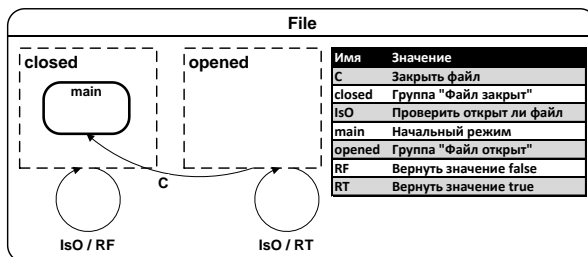
3. Плагин Acts as State Machine [5] для Ruby on Rails позволяет описать логику веб-приложения с использованием автоматной парадигмы программирования. Этот плагин обладает простым синтаксисом, однако в нём не было реализовано наследование и вложение автоматных классов.

В данной работе устраняются недостатки приведённых выше решений. Для этого реализованы два подхода к переносу диаграмм переходов в программный код. Первый подход будет использовать объектно-ориентированные свойства языка Ruby, второй – динамические.

В качестве примера будут реализованы автоматные классы для чтения из файла и записи в файл, поведение которых может быть обобщено и структурировано с помощью наследования. Эти классы образуют иерархию, показанную на рис. 1.



**Рис. 1. Иерархия классов доступа к файлу**



**Рис. 2. Абстрактный класс работы с файлом**

### 3. Особенности языка Ruby

Ruby – кросс-платформенный, объектно-ориентированный динамический язык программирования с элементами функционального стиля [6]. Эти свойства языка Ruby позволяют:

1. Получить автомат, код которого удобен для чтения и дальнейшего сопровождения, благодаря возможности разработки предметно-ориентированного языка (DSL). Преимуществами применения DSL являются: возможность создания решения в терминах предметной области, таким образом, эксперты данной предметной области могут понимать, проверять и модифицировать написанный код; самодокументированный код; повышение качества, надёжности и сопровождаемости программного обеспечения.

2. Легко верифицировать автомат. Для этого потребуется решить две задачи: доказать корректность построения структуры автомата с помощью DSL и верифицировать автомат, применяя темпоральную логику. Так как свойством функциональных языков программирования является отсутствие побочных эффектов, то для доказательства корректности построения структуры автомата с

помощью DSL будет достаточно убедиться в корректной работе каждой отдельной функции.

3. Решить вопрос изоморфного переноса графической нотации в программный код [8] – при реализации группового перехода не потребуется дублирование кода перехода для вложенных состояний.

4. **Наследование и вложение автоматов с использованием объектно-ориентированных особенностей языка Ruby**

Используя графическую нотацию [2], на рис. 2

представлен абстрактный автоматный класс для работы с файлом (File) Определены общие для всех файлов переходы:

- IsO – позволяет определить, открыт ли в данный момент файл;
- C – закрывает файл.

Диаграмма поведения классов ReadFile и WriteFile, построенная с использованием наследования, приведена на рис. 3.

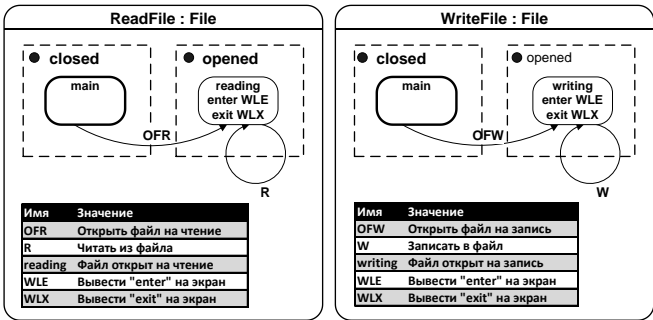


Рис. 3. Реализуемые классы ReadFile и WriteFile

Корневым элементом предлагаемой иерархии является абстрактный класс File, обобщающий некоторые аспекты доступа к файлу.

Автомат ReadFile наследует от автомата File группы closed (содержит состояние main) и opened (не содержит ни одного состояния). В группу opened добавляется новое состояние Reading, с действием при входе WLE и действием при выходе WLX. Добавляются новые переходы OFR (из состояния Main в состояние Reading) и R (петля для состояния Reading). Аналогичным образом строится автомат WriteFile наследованный от автомата File.

Реализуем приведённые выше автоматные классы, используя объектно-ориентированные особенности языка Ruby. При этом каждый автоматный класс и состояние являются отдельным классом языка.

Разработаем абстрактный автоматный класс `File`. Для этого предварительно реализуем все его состояния и вложенные группы состояний. Приведённый ниже класс отвечает за создание состояния `Main`, которое наследуется от класса абстрактного состояния `State`.



Созданное состояние имеет имя Main:

```
class Main < State
  def initialize container
    super :Main, container
  end
end
```

Разработаем группу Opened. Она имеет переход C в состояние Main с действием при переходе RT (рис. 2). Переходы задаются публичными методами (например, переход C), тогда как действия задаются приватными методами (например, действие RT):

```
class Opened < Automaton
  def initialize name, container=nil
    super name, container
  end
  def C
    @container.state :Closed
    RT()
  end

  private
  def RT; true; end
end
```

Аналогичным образом создадим вложенную группу Closed. После чего создадим автомат File, который будет включать в себя разработанные выше вложенные группы Opened и Closed.

```
class File < Automaton
  def initialize name, container=nil
    super name, container
    automaton Closed.new(:Closed, self),
      Opened.new(:Opened, self)
    initial :Closed
  end
end
```

Далее создадим автомат ReadFile (рис. 3) наследуемый от автомата File, для этого разработаем новое состояние Reading:

```

class Reading < State
  def initialize container
    super :Reading, container
  end
  def R; @container.state :Reading; end
end

```

Группа ReadOpened наследована от группы Opened. Следовательно, она будет иметь все переходы и состояния, которые были объявлены в группе Opened. Однако в эту группу требуется добавить состояние Reading, что достигается вызовом метода automaton – данный метод используется для создания вложенной группы с заданными состояниями.

Начальным состоянием вложенной группы ReadOpened будет состояние Reading, что достигается вызовом метода initial.

```

class ReadOpened < Opened
  def initialize name, container=nil
    super name, container
    automaton Reading.new(self)
    initial :Reading
  end
end

```

Аналогичным образом создадим состояние ReadMain, группу ReadClosed и автоматы ReadFile и WriteFile (рис. 3). В результате был создан программный код, отображающий автоматы, представленные на рис. 2, 3.

Можно отметить следующее достоинство использования объектно-ориентированного подхода к реализации наследования и вложения автоматов [4]: каждое состояние и автомат является отдельным классом, что позволяет сохранить иерархию автомата при переносе его в объектно-ориентированный код.

Недостаток данного подхода: синтаксическая избыточность, что не позволяет легко модифицировать логику переходов и расширять код.

В работе [2] графическая нотация может быть отображена в программный код лишь одним способом: с помощью объектно-ориентированного программирования, тогда как динамические языки программирования позволяют применить ещё и второй способ, основанный на метапрограммировании, который будет рассмотрен ниже.

## 5. Наследование и вложение автоматов с помощью динамических свойств языка Ruby

Этот подход устраняет синтаксическую избыточность, которая была отмечена в качестве недостатка предыдущего решения. В результате применения динамического языка Ruby был разработан специальный предметно-ориентированный язык, который состоит из трёх основных конструкций:

`current` – устанавливает начальное состояние автомата.

- `state` – создаёт новое состояние, которое может иметь несколько ключевых параметров:
- `:enter` – действие при входе в состояние;
- `:exit` – действие при выходе из состояния.
- `transition` – создаёт переход из одного состояния в другое, который может иметь следующие ключевые параметры:
- `:from` – состояние, из которого происходит переход;
- `:to` – состояние, в которое происходит переход;
- `:guard` – условие, при котором произойдёт переход;
- `:event` – название перехода;
- `:proc` – действие на переходе.

Ключевые параметры `:enter`, `:exit` и `:proc` являются лямбда-функциями. Условие `:guard`, при котором выполняется переход, является лямбда-предикатом – лямбда-функцией, которая возвращает значение «истина» или «ложь».

Приведем код, который реализует автоматы, изображённые на рис. 2, 3:

```
class AbstractFile < Automaton::Base
  def RT; lambda { true } end
  def RF; lambda { false } end
  def IsO; lambda { @file.is_opened? } end

  current :main

  state :closed,
    :enter => lambda {{ :current_state=>:main }}
  state :opened
  state :main

  transition :from => :opened, :to => :main,
    :event => :C
  transition :from => :closed, :to => :closed,
```

```

        :event => :IsO,
        :proc => RF()
    transition :from => :opened, :to => :opened,
        :event => :IsO,
        :proc => RT()
end

class ReadFile < AbstractFile
  def WLE; lambda { puts "enter" } end
  def WLX; lambda { puts "exit" } end
  def R; lambda { @file.read } end
  def OFR; lambda { |name| @file.open(name, "r") } end

  state :reading,
    :enter => WLE(),
    :exit => WLX()
  state :opened,
    :enter => lambda { { :current_state=>:reading } }

  transition :from => :reading, :to => :reading,
    :event => :R
  transition :from => :main, :to => :reading,
    :event => :OFR
end

```

Рассмотрим конструкции предметно-ориентированного языка, использованные в этой программе:

- установить начальное состояние автомата в main:

```
current :main
```

- создать новое состояние с именем writing. При входе в данное состояние выполняется функция WLE(). При выходе из данного состояния выполняется функция WLX():

```

state :writing,
  :enter => WLE(),
  :exit => WLX()

```

- создать переход из состояния closed в состояние closed, что образует петлю при входном воздействии IsO(). В случае перехода будет выполнена функция RF():

```

transition :from => :closed, :to => :closed,
  :event => :IsO,
  :proc => RF()

```

Рассмотрим наследование автомата ReadFile. Для этого необходимо добавить новое состояние Reading, которое будет являться основным для группы Opened, и два новых перехода R и OFR:

```
class ReadFile < AbstractFile
  state :reading,
    :enter => WLE(),
    :exit => WLX()
  state :opened,
    :enter => lambda {{:current_state=>:reading}}

  transition :from => :reading, :to => :reading,
    :event => :R
  transition :from => :main, :to => :reading,
    :event => :OFR
end
```

Таким образом, преимуществом данного подхода является отсутствие синтаксической избыточности, что позволяет использовать такие преимущества DSL, как самодокументированный код, простота понимания и расширяемость.

В тоже время недостатком данного подхода является потеря иерархии родительского автомата: состояние автомата не является отдельным классом, как было при использовании объектно-ориентированного подхода.

## 6. Протоколирование

Благодаря такому свойству динамических языков программирования, как метапрограммирование, можно разработать модуль, который будет автоматически отслеживать переходы, и выводить об этом соответствующую информацию. При этом не важно, был ли использован объектно-ориентированный или динамический подход к реализации автоматов.

Рассмотрим пример для автомата ReadFile:

```
class ReadFile < AbstractFile
  def initialize name, container=nil
    super name, container
    automaton ReadOpened.new(:Opened, self),
      ReadClosed.new(:Closed, self)
  end
end
```

```
tracer  
end
```

Данный класс отличается от того, который был использован в разделе 7 лишь одной строкой `tracer`. Это макровывоз, который для каждого перехода и состояния выводит отладочную информацию, например:

```
ReadMain::OFR begin  
In the OFR  
ReadMain::OFR end
```

## 7. Заключение

В настоящей работе были разработаны два подхода, которые позволяют изоморфно переносить диаграммы состояния в диаграммы поведения, а после этого далее изоморфно преобразовывать полученный результат в программный код. При этом были рассмотрены объектно-ориентированные и динамические свойства языка программирования Ruby.

Достоинством использования объектно-ориентированного подхода к реализации наследования и вложения автоматов [2] является сохранение иерархии автомата при переносе его в объектно-ориентированный код благодаря тому, что каждое состояние и автомат являются отдельным классом.

Однако данный подход обладает таким недостатком, как синтаксическая избыточность и не позволяет легко модифицировать логику переходов.

Показано, что для динамических языков программирования может быть разработан предметно-ориентированный язык (DSL), который позволяет:

- создать решение в терминах предметной области. В результате эксперты в этой области могут понимать, проверять и модифицировать написанный код;
- обеспечить самодокументированный код.

Недостатком этого подхода является потеря иерархии родительского автомата, так как состояние и автомат не являются отдельными классами, как было реализовано в объектно-ориентированном подходе.

Таким образом, оптимальным решением будет являться объединение объектно-ориентированного и динамического подходов: предметно-ориентированный язык будет использован для создания классов объектно-ориентированного подхода.

## 8. Список литературы

- [1] Поликарпова Н. И., Шалыто А. А. Автоматное программирование. Учебно-методическое пособие. СПбГУ ИТМО. 2007. <http://is.ifmo.ru/books/ umk.pdf>
- [2] Шопырин Д. Г., Шалыто А. А. Графическая нотация наследования автоматных классов // Программирование. 2007. № 5, с. 62–74. [http://is.ifmo.ru/works/ 12\\_12\\_2007\\_shopyrin.pdf](http://is.ifmo.ru/works/ 12_12_2007_shopyrin.pdf)
- [3] Степанов О. Г., Шалыто А. А. Предметно-ориентированный язык автоматного программирования на базе динамического языка Ruby // Информационно-управляющие системы. 2007. № 4, с. 22–27. [http://is.ifmo.ru/works/ 2007\\_10\\_05\\_aut\\_lang.pdf](http://is.ifmo.ru/works/ 2007_10_05_aut_lang.pdf)
- [4] Астафуров А. А., Шалыто А. А. Декларативный подход к вложению и наследованию автоматных классов при использовании императивных языков программирования. // Software Conference (Russia). М.: ТЕКАМА. 2007, с. 230–238. [http://is.ifmo.ru/works/ astafurov\\_secr\\_word\\_2003.pdf](http://is.ifmo.ru/works/ astafurov_secr_word_2003.pdf)
- [5] Scott B. Acts As State Machine. [http://agilewebdevelopment.com/plugins/acts\\_as\\_state\\_machine](http://agilewebdevelopment.com/plugins/acts_as_state_machine)
- [6] Thomas D., Fowler C., Hunt A. Programming Ruby. Second Edition. Texas: Pragmatic Bookshelf. 2004
- [7] Parr T. The Definitive Antlr Reference: Building Domain-Specific Languages. Texas: Pragmatic Bookshelf. 2007
- [8] Заякин Е. А., Шалыто А. А. Метод устранения повторных фрагментов кода при реализации конечных автоматов. СПбГУ ИТМО. 2007. [http://is.ifmo.ru/projects/life\\_app/](http://is.ifmo.ru/projects/life_app/)

# **Analyst in Agile — archaism or need?**

**Andrey Bibichev**  
**Customized InformSystems Ltd.**  
**email: [andrew@custis.ru](mailto:andrew@custis.ru)**

## **Abstract**

Among the issues of applicability and use of the Agile methodologies there are three most popular: in what measure are they compatible with fixed-price contracts; what is the managers' role; what about analysts. The first two issues are covered by a number of reports and articles from gurus of Agile movement. But the last one is not covered so well, and by this report, we make an attempt to improve situation.

First part discusses the reasons why there is an opinion that there is no place for analyst in Agile. The main one is a misinterpretation of some theses and "slogans" which are found in descriptions of Agile methodologies. For example, cross-functionality of team members, denial of need for the detailed specifications, desire to escape the unnecessary documentation, etc.

Second part describes the problems emerging at different stages of the Agile development process, which are usually hard to solve without an analyst: formation of the domain model; specifying tasks for an iteration; interaction with business-staff (domain expertise, clarifying the requirements' details); quality assurance; pilot deployment and initial support of software.

Third part gives a detailed description and analysis of the options for interaction between analyst(s) and developer team: analyst inside the team; Product Owner acting as analyst; analyst being the Product Owner's assistant; separate department of analysts. We discuss the benefits and disadvantages of each option, and conditions under which it can be applied.

Final part briefly addresses related issues, such as: what is Agile specifications; where, how and why to deal with an internal project documentation; why domain models are again in mainstream; how to bring up an analyst.

All materials are based on the practical use of Scrum methodology in Russian software company, and are correlated with authoritative views and statements.

**Keywords:** Agile; Scrum; Analyst; Product Owner; Domain Model.



# Аналитик в Agile — архаизм или необходимость?

Бибичев Андрей  
ООО «Заказные ИнформСистемы»  
email: andrew@custis.ru

## Тезисы

Среди вопросов о применимости и использовании Agile-методологий можно выделить три наиболее популярных: совместимость с проектами фиксированной стоимости; какова роль менеджеров; как быть с аналитиками. Что касается первых двух, то им уже посвящен целый ряд докладов и статей от гурзу Agile-движения. А вот последний вопрос освещен не так здорово, что и пытается поправить данный доклад.

В первой части обсуждаются причины, по которым возникает мнение, что в Agile аналитикам не место. Основная из них — неверная (гипертрофированная) интерпретация некоторых тезисов и «слоганов» из описаний Agile-методологий, таких как кроссфункциональность членов команды, отрицание необходимости в подробных спецификациях, стремление к избавлению от создания ненужной документации и т.д.

Вторая часть посвящена описанию задач, которые возникают на разных этапах Agile-процесса разработки и при решении которых роль аналитика почти незаменима: формирование модели предметной области; формулировка задач на итерацию; взаимодействие с представителями бизнеса (экспертиза в предметной области, прояснение деталей постановки); контроль качества; пилотное внедрение и начальное сопровождение программного обеспечения.

В третьей части подробно рассматриваются и анализируются возможные варианты взаимодействия аналитика(ов) и команды разработчиков: аналитик внутри команды; Product Owner, исполняющий функции аналитика; аналитик — помощник Product Owner-а; обособленный отдел аналитиков. Обсуждаются плюсы и минусы каждой из схем, условия применимости.

Заключительная часть коротко затрагивает такие смежные вопросы как: что такое Agile-постановки; где, как и зачем вести внутреннюю проектную документацию; почему модели предметной области снова в моде; как вырачивать аналитиков.

Весь материал основан на практическом опыте внедрения и использования Agile-методологии Scrum в российской компании-разработчике программного обеспечения enterprise-

уровня, а также соотнесен с авторитетными мнениями и высказываниями.

**Keywords:** гибкие методологии; Scrum; аналитик; Product Owner; модель предметной области.

## **1. Введение**

Гибкие методологии набирают популярность в России и странах СНГ. Как правило, при рассмотрении возможности перехода на них (и даже в процессе перехода или начального использования), почти неизбежно встанет целый ряд вопросов, если не сказать, сомнений. Наиболее частые из них:

1. Как быть с fix-price контрактами в Agile?
2. Какова роль менеджера в Agile, и как эта роль соотносится с понятием Product Owner?
3. Нужны ли аналитики в Agile, и если да, то как должно быть организовано взаимодействие с ними?

Что касается ответов на первые два вопроса, то ответы на них можно найти в великолепных презентациях двух главных популяризаторов методологии Scrum — Джефа Сазерленда [1] и Хенрика Книберга [2]. Кроме того, весьма полезным в этом смысле может оказаться и знакомство с докладом Стаси Бродерик [3], в котором она объясняет, как перевернуть с головы на ноги классический треугольник сроки/объем/бюджет.

А вот найти готовый ответ на третий вопрос, как ни странно, не так-то просто.

Данный доклад призван попытаться хотя бы частично заполнить этот пробел.

## **2. Необходимые оговорки**

### **2.1. Бизнес- и системные аналитики**

Достаточно распространенным является деление аналитиков на бизнес-аналитиков и системных аналитиков. Первые больше концентрируются на вопросах и тонкостях функционирования автоматизируемого бизнеса, являются экспертами в предметной области, но далеки от каких-либо системных «заморочек». А вторые — на системном анализе, построении информационных моделей, имеют представление об ограничениях и

особенностях используемых технологий, участвуют в архитектурном дизайне.

В данном докладе столь четкого деления не используется и везде употребляется термин «аналитик» без дополнительных уточнений. При этом в большинстве случаев предполагается, что это, скорее, бизнес-аналитик, но которому не чужды системное мышление и примерные представления о технологических особенностях разрабатываемой системы, т.е. кроссфункциональный аналитик (что вполне в духе Agile) с уклоном в бизнес-область.

## **2.2. «Мы говорим Agile, подразумеваем Scrum»**

Сейчас методология Scrum настолько популярна и распространена, что в большинстве современных докладов и статей, посвященных Agile, как правило, основное внимание уделяется именно Scrum. Это похоже на доминирование Internet Explorer-a, которое было на рынке Web-браузеров несколько лет назад.

Перефразируя известный лозунг о партии и Ленине: «Мы говорим Agile, подразумеваем Scrum! Мы говорим Scrum — подразумеваем Agile!»

Данный доклад не является исключением из этой тенденции — его материал предполагает использование именно Scrum в качестве Agile-методологии. Но следует понимать, что это не является серьезным ограничением на применимость описанных подходов в сочетании с другими гибкими методологиями — такими как eXtreme Programming (XP), Feature Driven Development (FDD), Open Unified Process (OpenUP) и т.п. — изменятся только терминология и некоторые несущественные нюансы.

Читателя, недостаточно знакомого с методологией Scrum, хочется адресовать к широко известному в узких кругах практическому введению Хенрика Книберга «Scrum и XP с передовой» [4].

## **2.3. На чем основан доклад**

В основу материала легли отнюдь не виртуальные эксперименты или философские разговоры в курилке, а реальный опыт перевода целой компании на методологию

Scrum. Подробная информация об этом опыте была представлена в докладе на конференции РИТ-2008 [5].

### 3. Мифы Agile

Нам не дано предугадать,  
Как слово наше отзовется.

Эти две строки из стихотворения Тютчева как нельзя лучше характеризуют ситуацию вокруг многих тезисов и принципов, декларируемых как в рамках Agile в общем, так и в рамках отдельных представителей гибких методологий, таких как Scrum, XP или Lean. Вот некоторые из них:

- ✓ наиболее эффективные команды — это кроссфункциональные команды;
- ✓ вначале итерации команде не должно требоваться наличие подробных спецификаций для решения поставленных задач;
- ✓ не следует создавать «лишние» артефакты (например, write-only документацию): работающий программный код — единственный артефакт, необходимость которого не вызывает сомнений (а XP почти полностью отрицает необходимость ведения внутренней проектной документации);
- ✓ у разработчиков должна быть возможность общаться с пользователями и представителями бизнеса напрямую, т.е. без лишних посредников (а в XP еще требуется присутствие представителя заказчика непосредственно в команде разработчиков).

Список можно продолжать, но даже этого достаточно, чтобы мнение типа «аналитику нет места в Agile, либо он должен уметь программировать» получило столь широкое распространение, что с этим уже приходится считаться.

Очень часто приходится слышать вопросы:

- И что, при переходе на Scrum нам нужно учить аналитиков программировать?
- Product Owner — это руководитель проекта, выполняющий еще и функции аналитика?
- Теперь вообще не следует предварительно прорабатывать задачу или требование перед постановкой их реализации в итерацию?

Давайте попробуем разобраться по порядку:

### 3.1. Кроссфункциональность

По сути, это тот горизонт, к которому, по возможности (и в рамках разумного), следует стремиться, но который вряд ли когда-либо будет достигнут.

Когда идеологи Agile говорят о кроссфункциональности, нужно понимать, чему они ее противопоставляют.

На рис. 1 приведены результаты исследования зависимости условной комфортности командной работы от количества ролей в команде [1]. Видно, что большинство экспериментальных точек лежит в окрестности 20 ролей! Двадцати! При этом весьма благоприятные условия наступают уже при количестве ролей, меньшем 10. А если в команде около 5 ролей — то это почти безграничное счастье каждого члена команды.

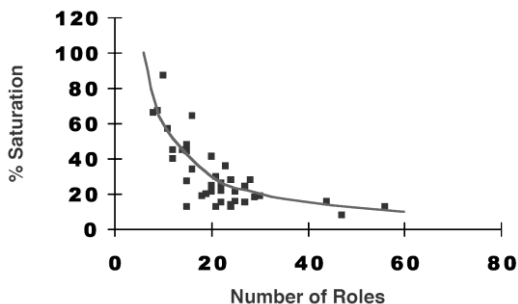


Рис. 1. Количество ролей в команде

Так что пропаганда кроссфункциональности — это попытка объяснить и донести до широких слоев, что не должно быть избыточного или искусственного деления по ролям, хотя при этом наличие разумной специализации и различия в квалификации вовсе не отрицаются.

Здесь весьма важными являются выделенные слова: например, если требуется участие аналитика в ручном тестировании пользовательского интерфейса, то аналитик не должен от этого отказываться только на том основании, что это не входит в его обязанности.

Помните о золотой середине!

### **3.2. Нет подробным спецификациям**

Это противопоставление водопаду (waterfall) [6] и тяжеловесным (heavy-weight) процессам, в которых сильна вера в магию подробных технических заданий и формальных спецификаций на компоненты системы: умные люди долго думают и тщательно пишут хорошие и подробные бумажки, после дело остается за малым — перевести это всё в машинный код при помощи труда кодеров (даже не программистов, и уж тем более не кроссфункциональных разработчиков). Эта магия настолько часто не срабатывает, что в большинстве случаев даже в успешных проектах разработчики (если таковые оказываются среди кодеров) почти не используют эти технические задания и спецификации или настолько отходят от них (для того чтобы система оказалась жизнеспособной), что ценность этих самых вымученных документов сводится к нулю.

В спецификациях есть еще одна уязвимость: чем более подробными и/или низкоуровневыми они оказываются, тем меньше шансов, что заказчик или потенциальный пользователь смогут их прочитать, причем о понимании и верификации, как правило, и речи нет.

Однако, это не повод совсем не прорабатывать задачу или требование перед тем, как ставить их в итерацию. Опять же, важен разумный компромисс: нужно проработать на столько, чтобы команда в рамках итерации могла сконцентрироваться на реализации соответствующей функциональности, а не бесконечных выяснениях «а что же, собственно, нужно», при этом могут произойти какие-то уточнения и даже небольшие изменения в начальном **видении** (vision) задачи.

Более детально данный вопрос обсуждается в последней части доклада.

### **3.3. Минимум документации**

С развитием интернета и блогосферы всё больше ощущается проблема не с писателями, а с читателями — пишут миллионы, а читают единицы. С проектной документацией во многом часто та же ситуация — если её много, то вряд ли её кто осилит целиком, а если и найдутся уникамы, то весьма сомнительно,

что они смогут всё должным образом усвоить, отличив при этом главное от второстепенного.

Кроме того, проектная документация имеет тенденцию к очень быстрому устареванию, если не сказать «протуханию» — в особенности в сочетании с Agile, где изменения — норма, а не форс-мажор. Это усугубляется тем обстоятельством, что консистентность документации, в отличие от работающей системы, очень тяжело проверить.

В связи с этим, в Agile-методологиях всячески советуется минимизировать документацию:

- ✓ избегать write-only документации, т.е. той, которую заведомо никто не прочитает;
- ✓ писать лаконично, выделяя главное и опуская излишние детали, ибо детали меняются чаще;
- ✓ использовать больше визуальных образов, схем, графических нотаций.

Но это вовсе не значит, что минимизировать надо все и до нуля. При полном отсутствии документации, скорее всего, возникнут следующие проблемы:

- ✓ тяжело вводить новых людей в курс проекта;
- ✓ велика вероятность утери общей концепции и видения (как проекта в целом, так и отдельных его частей);
- ✓ сложно осуществлять контроль качества, так как не понятно с чем сверять (в особенности это касается полнофункционального регрессионного тестирования, которое полностью автоматизировать очень тяжело);
- ✓ сложно сопровождать и развивать старую функциональность, так как все уже забыли почему и зачем именно так сделали;
- ✓ и т.д.

Если они не возникли, считайте, что вам повезло. Но если вы думаете о будущем проекта — лучше не полагаться на везение.

### **3.4. Общение разработчиков с пользователями**

Такую возможность действительно нужно обеспечить, т.к. это бывает очень полезно:

- ✓ повышает качество (меньше искажений в передаче информации, выше мотивация при прямом контакте с пользователями);
- ✓ выше вероятность сделать то, что нужно, а не то, что просили («спасибо, вы сделали то, что я просил, но это не то, что мне нужно»);
- ✓ ускоряет процесс (короче цепочка передачи информации).

Как обычно, в этой идиллии есть существенные «но». Во-первых, разработчики больше обращают внимание на техническую шелуху, а пользователи и представители заказчика — на бизнес-шелуху, из-за чего им иногда тяжело договориться, а зачастую и понять друг друга. Во-вторых, очень часто встречается ситуация, которую принято описывать термином «вязкий заказчик»: на уточнение и прояснение требований приходится тратить большое количество усилий и времени. Как правило, это связано с большой бюрократизацией у заказчика или с экстремально высокой занятостью ключевого персонала у заказчика, а ведь всякие нетривиальные моменты и важные решения приходится выяснять и согласовывать именно с ключевыми фигурами в бизнес-процессе, а не линейными исполнителями, которые редко когда видят дальше своих повседневных рутинных обязанностей.

В связи с этим возникает идея, чтобы был кто-то, помогающий разработчикам и пользователям находить общий язык и договариваться, а кроме того, преодолевать вязкость среды своими личными усилиями и навыками.

## **4. Возможные функции аналитика**

Итак, в Agile нет антагонизма к роли аналитика, как таковой. Но действительно ли аналитик может быть полезен и в чем именно?

### **4.1. Связующее звено между разработчиками и заказчиками**

В отличие от классической интерпретации функций аналитика, в Agile именно обеспечение эффективной связи между заказчиками (пользователями) и командой разработчиков играет, по сути, ключевую роль (см. рис. 2). Основные причины описаны в разделе 3.4.





**Рис. 2. Элементы мозаики**

Т.е. аналитик оказывается тем парнем (на самом деле, чаще девушкой), которому(ой) доверяют и пользователи, и разработчики:

- ✓ Если возникают проблемы в формулировании или интерпретации требований, то необходимо, чтобы кто-то организовал общее совещание, на котором бы оказались все вовлеченные стороны (и от разработки, и от бизнеса), при этом нужно управлять ходом дискуссии, помочь сформировать общее решение и сформулировать его таким образом, чтобы оно было понятно всем заинтересованным лицам.
- ✓ Если пользователи страстно хотят одного, а разработчики упрямо настаивают на другом, то кто-то должен помочь найти компромисс или убедить разработчиков сделать так, как просят заказчики и пользователи.
- ✓ Если для решения задачи требуется прояснить ряд существенных деталей, а представители заказчика уходят в тину или устраивают круговую поруку (один ссылается на другого, тот на третьего и так далее, пока не замкнется в круг) или противоречат друг другу, то нужен кто-то, кто сможет эффективно выйти из этой ситуации.
- ✓ Если заказчики активно используют свой внутренний жаргон (бизнес), а разработчики — свой (технический), то нужен кто-то, кто сможет быть переводчиком.

В большинстве случаев этот кто-то — это аналитик, которому помогают менеджеры, когда требуется административный ресурс, и ключевые эксперты проекта или даже компании, когда требуется генерация или верификация нестандартных решений.

#### **4.2. Экспертиза в предметной области**

Это достаточно стандартная функция для бизнес-аналитиков. Даже если у заказчика есть внятные эксперты в предметной области, умеющие доходчиво изложить свои знания, полезно иметь в команде (или «рядом» с ней) кого-то, кто будет накапливать эту экспертизу. Это позволит не зависеть от внешних экспертов при работе с другими заказчиками на том же вертикальном рынке или при переходе от штучного заказного решения к тиражируемому (коробочному), ведь как известно, зачастую покупается не продукт, а экспертиза и опыт.

#### **4.3. Систематизация и построение моделей**

В Agile задачи систематизации представлений и требований, построение моделей предметной области и отдельных элементов системы принято решать при помощи группового интеллекта — на общекомандных дизайн-сессиях, сессиях планирования и т.п. И это оказывается очень эффективным — во время таких общих обсуждений рождаются наиболее адекватные модели и качественный дизайн. Поэтому аналитик в Agile не обязан уметь делать это самостоятельно, но практика показывает, что если аналитик способен предложить «начальное приближение», то скорость и эффективность общего обсуждения резко возрастают. В отличие от «водопада» такая ситуация достаточно комфортна — не страшно сделать ошибку или что-то не учесть — коллеги помогут это исправить.

Если же аналитик не обладает достаточным уровнем системности мышления или навыками моделирования (составления моделей предметной области), то эти функции возьмет на себя команда, возможно даже с привлечением экспертов из других проектов.

#### **4.4. Контроль качества**

Традиционно считается, что контролем качества занимаются специальные люди (или роботы?), которые выполняют приемочные испытания по описанным методикам и программам. В Agile говорят, что этого не достаточно и к этому, как минимум, прибавляют:

- ✓ регулярные, 1–2 раза в месяц, демонстрации представителям заказчика для получения неотложной обратной связи;
- ✓ unit-тесты для максимальной автоматизации и упрощения процесса регрессионного тестирования, локализации возникающих ошибок, формализации спецификаций на части системы до уровня проверяющего кода;
- ✓ непрерывную интеграцию для раннего обнаружения проблем.

Это всё здорово, полезно, действительно повышает качество и минимизирует некоторые риски. Однако кто проверит, что сделали то, что нужно, и что пользоваться удобно? Пользователи и заказчики на демонстрации — это, конечно, вариант, но, во-первых, не факт, что среди них окажутся заинтересованные именно в этой функциональности, а во-вторых, так можно потерять лицо (демонстрация превратиться в сессию тестирования и отладки, причем на глазах у изумленной публики).

Генрих Книберг в своей презентации «10 способов напорочить со Scrum и XP» [7] говорит о критерии «сделано» (DoD, Definition Of Done). Согласно этому критерию, задача не считается до конца выполненной до тех пор, пока соответствующая функциональность не пройдет функциональные тесты и пока Product Owner не согласится с тем, что сделано то, что нужно. Т.е. получается, что Product Owner участвует в контроле качества — когда команда считает, что задача успешно решена, в этом должен еще убедиться и Product Owner.

Достаточно очевидно, что вместо (или совместно с) Product Owner-а (ом) эта почетная обязанность может быть возложена на аналитика.

При этом лучше не втягивать аналитиков в ручное регрессионное тестирование или другие рутинные процедуры — только в проверку правильности реализации новой функциональности.

#### **4.5. Участие в пилотных внедрениях**

Во время первых, «пилотных» внедрений системы, когда процедура еще не отлажена до формальных регламентов, возникает много проблем и нетривиальных трудностей:

- ✓ обучение пользователей при нехватке или даже отсутствии учебных материалов и инструкций;
- ✓ помощь пользователям в освоении системы («живая справка»);
- ✓ исправление ошибок, возникших в результате некорректных действий неопытных пользователей или сбоев в системе;
- ✓ фиксация всех узких мест, дополнительных пожеланий и требований, ошибок и неточностей (дабы донести до разработчиков и Product Owner-a);
- ✓ перегрузка и выверка данных (в особенности справочников);
- ✓ начальная настройка системы.

Привлечение аналитика к их решению существенно помогает, так как это человек, который хорошо знаком не только с пользователями и характером их работы, но и с самой системой.

#### **4.6. VIP-сопровождение**

Если у вас есть VIP-клиенты (крупные заказчики, первые внедрения и т.п.), то к процессу сопровождения и внедрения почти неизбежно придется привлекать высококвалифицированный персонал. Ведь у таких клиентов и проблемы нестандартные, и пожелания сложные, а bug-репорты зачастую выливаются в новый пласт функциональности или объемные доработки — да, грань между feature request и bug-ом в подобной ситуации почти условна.

И здесь «опять на помощь приходят они»: те, кто хорошо знает и бизнес, и систему и к тому же умеет понимать пользователей и объяснять разработчикам.

#### **5. Схемы взаимодействия аналитик-команда**

В Agile большое внимание уделяется командной работе, самоорганизации команды (уход от микро-менеджмента). Как

организовать взаимодействие аналитика с командой с учетом возлагаемых на него функций? Однозначного ответа на этот вопрос нет. Существуют разные варианты, причем в зависимости от обстоятельств эффективными оказываются те или иные. Ниже рассмотрены те схемы, с которыми приходилось сталкиваться на личном опыте.

### **5.1. Product Owner — аналитик**

Это самый простой и очевидный случай. Product Owner отвечает за продукт, за сбор и приоритезацию требований, является своеобразным представителем заказчика, но на стороне исполнителя, отвечает или помогает ответить на уточняющие вопросы. Всё это тесно переплетается с функциями аналитика, обсуждаемыми выше.

Так что можно решить, что функции аналитика выполняет Product Owner. Или, если угодно, наоборот — аналитик исполняет роль Product Owner-а.

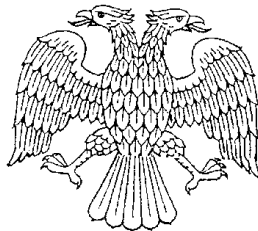
Среди плюсов такой схемы: простота, минималистичность, относительное удобство и для заказчика, и для разработчиков — и те и другие всегда знают к кому именно нужно обратиться со своими вопросами.

Недостатки:

- ✓ Получается, что слишком многое зависит от одного человека — большая нагрузка и связанные с этим риски «бутылочного горлышка».
- ✓ Экстремальная незаменимость — а если в отпуске, а если заболел (дай Бог ему здоровья!).
- ✓ Ситуация на рынке труда такова, что и аналитика не просто найти, и Product Owner-а тоже. А и то, и другое в одном лице — на порядки сложнее.
- ✓ Вряд ли получится плотно привлечь такого Product Owner-а к сопровождению системы или активному участию в пилотных внедрениях.
- ✓ Есть вероятность, что Product Owner будет откладывать решение каких-то задач не потому, что у них низкий приоритет, а потому, что он пока еще не успел как следует проработать постановку. Т.е. убивается вся идея приоритезации работ на основании потребностей бизнеса, а не исходя внутренних или технических обстоятельств.

## 5.2. Аналитик — помощник Product Owner-а

Большинство недостатков схемы 5.1 можно преодолеть, если поделить обязанности Product Owner-а и аналитика между двумя людьми (и это не чуждо нашей культуре — см. рис. 3). Это достаточно распространенная практика. Как правило, при этом главный решает, что в какую очередь делать, и дополнительно выполняет менеджерские функции (и/или отвечает за клиентские отношения). А помощник больше концентрируется на содержании и деталях работ, т.е. играет роль аналитика.



**Рис. 3. «Двуглавый» Product Owner**

На конференции QCon 2008 London в докладе «Agile Mashups» Рейчел Дэвис (Rachel Davies), ссылаясь на свою обширную практику консультанта, утверждала, что это самая распространенная схема.

Однако, и у нее всё же присутствуют недостатки:

- ✓ Деятельность аналитика недостаточно прозрачна для команды, поскольку аналитик в нее не входит.
- ✓ Велика вероятность, что команда будет воспринимать аналитика, как Product Owner-а (или даже руководителя), т.е. видеть в нем не помощника и эксперта, а начальника, что почти наверняка убьёт критичность восприятия предложений и моделей, предлагаемых аналитиком — их будут воспринимать не как начальное приближение и дополнительную информацию, а как инструкции к действию.
- ✓ Опять же, не так-то просто привлечь такого аналитика к сопровождению.

### **5.3. Аналитик внутри команды**

Можно пойти еще дальше и поместить аналитика внутрь команды. Что это значит:

- ✓ аналитик сидит вместе со всеми, т.е. в одной комнате с разработчиками;
- ✓ аналитик участвует в Scrum-митингах наравне с остальными (рассказывает, что делал вчера и что собирается делать сегодня);
- ✓ работа аналитика учитывается при планировании итерации;
- ✓ аналитик может «делиться» своей работой с другими (по общему согласию);
- ✓ и наоборот, аналитик может привлекаться к нехарактерным для себя работам, чтобы помочь остальной команде в непростую минуту — например, подготовить тестовые данные, прогнать часть ручных тестов.

Звучит здорово. И работает здорово! Однако бывают обстоятельства, при которых данная схема не подходит:

- ✓ одной предметной областью (или сильно похожими) занимается несколько команд разработчиков, так что держать по аналитику в каждой команде нет особого смысла;
- ✓ реализация больших или технически сложных проектов в методологии Scrum-of-Scrum;
- ✓ в проекте так много технических и технологических тонкостей и сложностей, что команда в основном сконцентрирована на них, а аналитик в такой команде оказывается инородным телом;
- ✓ нехватка квалифицированных аналитиков.

### **5.4. Внешний отдел аналитиков**

Это то, что больше характерно для водопадных и тяжеловесных методологий, но может работать и в Agile — например, в обстоятельствах, описанных в предыдущем пункте.

Из дополнительных плюсов: простота и естественность обмена информацией, навыками и практиками между аналитиками.

Основной недостаток: экстремально высокая отдаленность от команды разработчиков, что может привести к взаимному

недоверию. Кроме того, вырастает вероятность отката к прежним практикам и прежним проблемам.

## **6. Смежные вопросы**

В последней части доклада кратко рассматриваются такие смежные вопросы как:

- Что такое Agile-спецификации? В каком виде следует формулировать и оформлять задачи на итерацию?
- Внутренняя проектная документация: как и в чём вести, какой инструментарий использовать?
- Что такое Domain Driven Design? И почему модели предметной области снова в моде? Как это связано с популярностью Agile?
- Как выращивать аналитиков внутри компании?

## **7. Заключение**

Пожалуйста, не превращайте Agile в очередной карго-культ [8]! Пытайтесь осознать основные идеи и следовать им — попытки использовать материалы по Agile-методологиям, как исчерпывающие инструкции или догматы, приведут вас к чему угодно, кроме того, что изначально вкладывалось в понятие Agile.

Описанные варианты, способы взаимодействия и подходы к проблемам бизнес-анализа не являются исчерпывающими или строго обязательными — в вашем конкретном случае другие приемы могут оказаться более эффективными и оправданными. Было бы очень интересно узнать об альтернативах — не замалчивайте их!

## **References**

[1] H. Kniberg, “The Manager’s Role in Scrum”, Scrum Gathering, <http://www.crisp.se/henrik.kniberg>, Nov 14, 2007.

[2] J. Sutherland, “Money for Nothing and Your Change for Free: Agile Contracts”, Agile 2008, <http://jeffsutherland.com/scrum>, Aug 19, 2008.



- [3] S. Broderick, “Introduction to Agile for Traditional Project Managers”, Agile 2007, <http://www.infoq.com>, Jul 13, 2008.
- [4] H. Kniberg, Scrum and XP from the Trenches, InfoQ, 2007.
- [5] А. Бибичев, “Практика внедрения Scrum: трудности и пути их преодоления”, РИТ-2008, <http://www.scrumtrek.ru>, 15 апреля 2008.
- [6] Royce, Winston, “Managing the Development of Large Software Systems”, Proceedings of IEEE WESCON 26 (August), 1970.
- [7] H. Kniberg, “10 ways to screw up with Scrum and XP”, Agile 2008, <http://www.crisp.se/henrik.kniberg>, Aug 19, 2008.
- [8] From the Editor (March/April 2000), “Cargo Cult Software Engineering”, IEEE Software, May 24, 2008.

# Performance Prediction of Client-Server Systems by high-level Abstraction Models

Alexander Pastyak, Yana Rebrova, Vladimir Okulevich  
Alexander.Pastyak@siemens.com Yana.Rebrova@siemens.com  
Vladimir.Okulevich@siemens.com

## Abstract

Performance modeling and simulation offer powerful approaches to predict performance characteristics of software applications and in particular web applications. At the same time the usage of modeling techniques in real projects is still limited due to comprehensive structure of applications and impossibility to know internal structure of application's components needed to create a precise model. In our work we build and evaluate the high level abstraction models of Content Management System "Joomla" to estimate their ability to provide reliable predictions for different performance characteristics. To build the models we consider three formalisms: Layered Queuing Networks, Performance Evaluation Process Algebra and Queuing Petri Nets. Predictions obtained from models simulation have been compared with measurements of the real system by HP Load Runner toolkit. Results show that overall throughput of the system can be predicted with high level of accuracy while prediction of response time by the same models is less reliable.

**Keywords:** Software performance simulation, performance modeling, web based systems

## 1. Introduction

Performance simulation of client-server systems has been a topic of interest for software engineering for a long time. A lot of formalisms and supporting methods have been developed to describe this kind of systems and obtain performance characteristics from related models. Most important of them are: Queuing Networks, Stochastic Process Algebras and Stochastic Petri Nets. All of these techniques have been applied to client-server applications and in particular to web applications, that showed their powerful ability to predict system behavior from the performance point of view. However the software modeling still has not become a daily practice in product life-cycle due to the complexity of model creation. It is caused by comprehensive structure of real-life applications and "black-box" nature of the application's components. To investigate model ability to predict system properties without detail information on internal structure we selected three different modern performance description formalisms to describe Content Management System "Joomla" [13]. During our study we built three

different performance models of this system deployed in several configurations of distributed environment. These models have been calibrated and analyzed to obtain performance predictions, which have been compared with experimental results received by HP Load Runner package [7].

## **2. Related Work**

Black box performance modeling approaches are most promising ones for analysis of real-life computer systems. Successful applications of such approaches for performance modeling of self-tuning controllers for web servers, storage systems and parallel applications are shown in [3, 8, 10]. These papers are focused on the techniques to understand parameter space of black-box system. On the other hand we consider simple performance models with high level of abstraction and estimate their capabilities to predict performance characteristic of real-life web application.

## **3. Problem Statement**

The goal of our study is to analyze formalisms which allow estimating performance characteristics for the system with unknown internal structure. In particular we are interested in the following questions:

- How to map different system architecture entities to the model elements (build the models)?
- What is the method to calibrate the models?
- Are models capable to make reliable predictions for throughput characteristic?
- What kind of predictions can be obtained from the models for response time characteristic?

Our investigations of these issues are based on analysis of the experimental test system which has been modeled with help of selected formalisms.

## **4. Performance Modeling Formalisms**

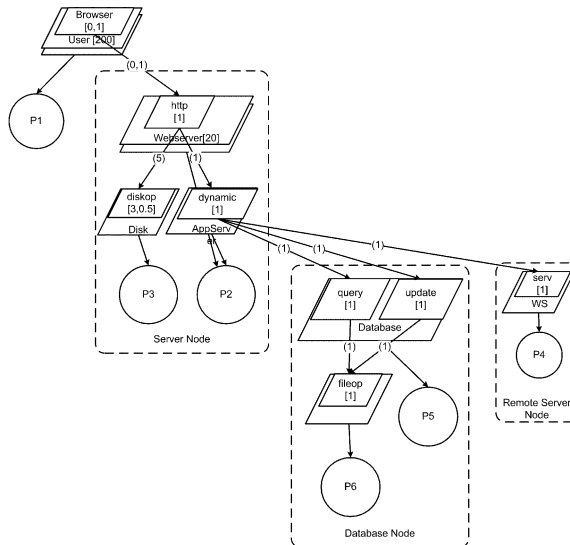
For our study we chose three formalisms by one from every group of modeling approaches: Queuing Networks (QN), Stochastic Process Algebras (SPA) and Stochastic Petri Nets (SPN). During this selection we were considering the following criteria:

- actuality: approach should be widely used for different performance investigations and being actively supported/developed;
- availability of tools: approach should be supported by easy-to-work tools.

Finally we decided to select Layered Queuing Networks (LQN) and LQNSolver tool [14] from QN. From SPA we chose Performance Evaluation Process Algebra (PEPA) and PEPA Workbench [15]. Also we opt for Queuing Petri Nets (QPN) and QPN Modeling Environment (QPME) [17] from SPN.

## 4.1 Layered Queuing Networks

Layered Queuing Networks (LQN) model [4, 5] is a canonical form for extended queuing networks with a layered structure. The layered structure arises from servers at one level sending requests to servers at lower levels as a consequence of request from a higher level. LQN is applied to any extended queuing network with multiple resource possession, in which multiple resources are held in a nested fashion. Resources are released in the reverse order of their acquisition and the resource order is consistent across the system. So, higher layer resources are acquired earlier and released later than in lower layers.



**Figure 1. Example of LQN Model**

Figure 1 illustrates the LQN notation with an example of a web server that has a connection to database and serves both static and dynamic pages. In LQN software resources are called tasks. The tasks have queues and provide classes of service which are called entries. In Figure 1 tasks are shown as parallelograms, containing nested parallelograms to describe entries. Processor resources are shown as circles attached to tasks which are using them. Stacked icons represent tasks or processors with multiplicity forming a multiserver. The multiserver may represent either a multi-threaded task, or a collection of identical users, or a symmetric multiprocessor with a common scheduler. Multiplicity is shown on the diagram with a label in brackets. For example, there are 20 copies of the task WebServer on Figure 1.

Entries have directed arcs to other entries at lower layers to represent service requests (requests may go directly through the layers). A request from one entry to another may return a reply to the original entry (a synchronous request) indicated on Figure 1 by solid arrows with closed arrowheads. For example, task AppServer send a request to task Database which then issue a request to task FileServer. While task FileServer is serving the request, tasks Database and AppServer are blocked. Alternatively a request may be forwarded to another entry for later reply, or may return no reply (an asynchronous request).

## 4.2 Performance Evaluation Process Algebra

In Performance Evaluation Process Algebra (PEPA) [6], system is considered as a set of components which carries out activities either individually or in cooperation with other components. Each activity is characterized by an action type  $\alpha$  and a duration  $r$  which is exponentially distributed. This is written as a pair  $(\alpha, r)$ . Duration may be any positive real number or it may be unspecified. Activity is called shared if several components synchronize over it. The distinguished symbol  $T$  is used to indicate that the rate is not specified by the component. Such component is said to be passive with respect to this action type. The rate of the shared activity is defined by cooperation with another component.

PEPA provides a set of combinators which allow building expressions to define behavior of components via activities. These combinators are presented below:

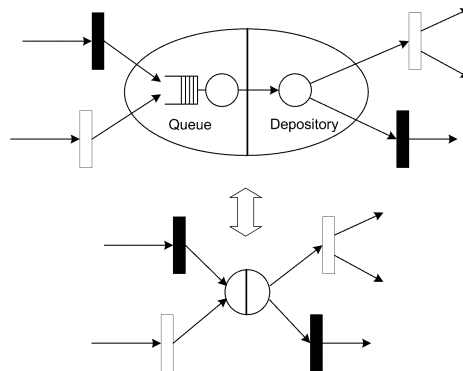
- **Prefix  $(\alpha, r).P$ :** Prefix is the basic mechanism by which behavior of components is constructed. This combinator implies that after the component has made activity  $(\alpha, r)$  it behaves as component  $P$ ;
- **Choice  $P1+P2$ :** This combinator represents a competition between components. The system may behave either as component  $P1$  or as  $P2$ . The probability to choose one of these components is defined by the rate of their first activity;
- **Cooperation  $P1<L>P2$ :** This describes the synchronization of components  $P1$  and  $P2$  over the activities in the cooperation set  $L$ . The components may proceed independently with activities whose types do not belong to this set. In cooperation, the rate of a shared activity is defined as the rate of the slowest component;
- **Hiding  $P/L$ :** This component behaves like  $P$  except that any activities of types within the set  $L$  are hidden, i.e. such an activity exhibits the unknown type  $\tau$  and the activity can be presented as an internal delay by the component. Such activity cannot be carried out in cooperation with any other component.
- **Constant  $A \text{ def } = P$ :** Constants are components whose meaning is given by a defining statement:  $A \text{ def } = P$  gives the constant  $A$  the behavior of the component  $P$ . This is how we assign names to components.

### 4.3 Queuing Petri Net

The main idea behind the Queuing Petri Net (QPN) modeling paradigm [1] is to add queuing and timing aspects to places of modeling formalism Colored Generalized Stochastic Petri Net (CGSPN) (subset of SPN) to provide means for direct representation of queuing disciplines. A place of CGSPN with an integrated queue is called a queuing place and consists of two components: the queue and a depository. This is depicted on Figure 2.

The behavior of the net is the following: tokens fired into a queuing place by any of its input transitions, are inserted into the queue according to the queue's scheduling strategy. After completion of its service, a token is immediately moved to the depository, where it becomes available for output transitions of the place. This type of queuing place is called timed queuing place. Also QPN introduces immediate queuing places (ordinary places) which allow pure scheduling aspects to be described. Tokens in ordinary places can be viewed as being served immediately. Scheduling in such places has higher priority than both scheduling/service in timed queuing places and firing of timed transitions. The rest of the net behaves like a normal CGSPN.

As it is shown in [2, 9] QPN has greater expressive modeling power than QN, extended QN and SPN. This formalism provides possibilities to model simultaneous resource possession, blocking synchronization and scheduling strategies.



**Figure 2. Structure of Queuing Place**

## 5. Architecture of the Test System

The following criteria have been considered during hardware and software selection for this study:

- hardware is to be widely available, represent different classes of systems and provide enough power to run appropriate applications;

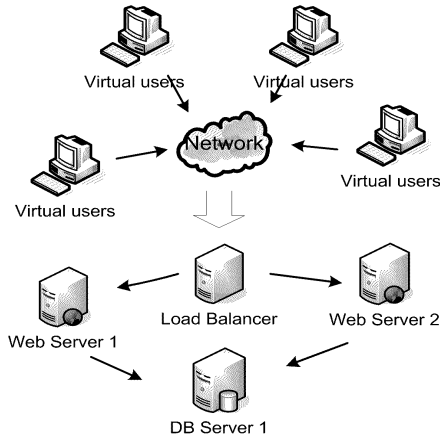
- software should be a common and popular web application with comprehensive internal structure, which hardly can be modeled in details.

For our test system we have chosen Content Managing System Joomla [13] deployed in the distributed environment with load balancer which is topologically placed in front of two web servers. [Figure 3]. Joomla is popular and convenient CMS used for many web sites. Being based on PHP and MySql it can be easily installed and configured in most of the environments.

Selected hardware and installed software are introduced in Table 1. All these hardware has been connected through 100Mbit Ethernet.

**Table 1. Hardware/Software for Test System**

<b>Title</b>	<b>Hardware</b>	<b>Software</b>
<b>Web Load Balancer</b>	2.16Ghz Core2Duo PC	Apache Web Server v2.2 [12] with enabled modules mod_proxy and mod_proxy_balancer
<b>Web Server 1</b>	3.5Ghz Celeron PC	Apache Web Server v2.2 with enabled module
<b>Web Server 2</b>	2.2 Ghz Core2Duo PC	mod_fast_cgi, PHP v5.2.6 [16] compiled with fast_cgi support; CMS Joomla, v1.5.3
<b>DB Server</b>	2.2 Ghz Core2Duo PC	MySQL v 5.1



**Figure 3. Test System Infrastructure**

Three different configurations have been analyzed in our study:

- Configuration 1: all client requests are processed by WebServer 1
- Configuration 2: all client requests are processed by Web Server 2
- Configuration 3: all client requests are sent to Web Load Balancer, which forwards them to Web Server 1 and Web Server 2.

To verify model predictions we have measured throughput and response time characteristics of the system in all these configurations by HP Load Runner [7]. This tool is widely used for load-testing of web applications.

## 6. Models of the System

To define the performance models for our system we separated the following entities, which have been later mapped to the appropriate model elements:

- Clients: independent tasks which send incoming requests to the Load Balancer. We used so-called closed workload where client can send a next request only when server has completed the previous one.
- Load Balancer: server which forwards requests to one of the Web Servers. The time for forwarding is negligible.
- WebServer 1 and WebServer 2: servers which run the Joomla installation and perform main part of request processing. Times to process single request by these servers are  $T_{WS1}$  and  $T_{WS2}$  respectively. For convenience in our models we use inverse values called service rates:  $Rate_{WS1} = T_{WS1}^{-1}$  and  $Rate_{WS2} = T_{WS2}^{-1}$ .
- Database: server which runs MySQL installation. Service rate for this server is  $Rate_{DB}$ .

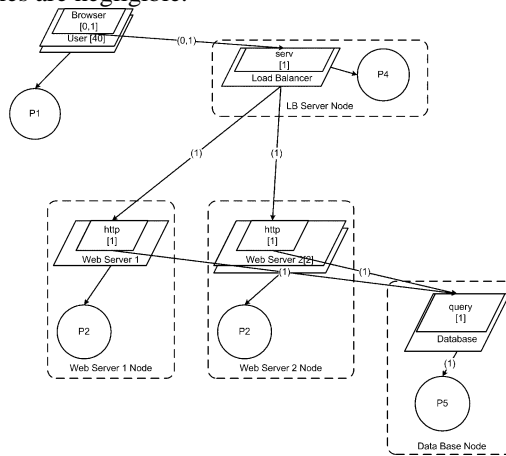
The rates have been defined through a normalization procedure described later. In the next sections we'll show how these entities are mapped to model elements.



## 6.1 LQN Model

In LQN model every entity derived above from the system architecture is mapped directly to a task element. Since tasks are running on processors and contain entries we have to define two more kinds of elements to complete the model. Graphical notation of the resulting model is shown on Figure 3. Every task is mapped to processor (circle) and contains entry which consumes processor time. Web Server 2 is shown with two tasks to reflect its Core2Duo CPU.

Time demands for WebServer 1, WebServer 2 and Database entries are defined as  $\text{Rate}_{\text{WS1}}^{-1}$ ,  $\text{Rate}_{\text{WS2}}^{-1}$ ,  $\text{Rate}_{\text{DB}}^{-1}$  respectively. The demands for Clients and Load Balancer entries are negligible.



**Figure 3. LQN Model of Test System**

## 6.2 PEPA Model

In case of PEPA model system entities are mapped to components. Every component runs infinitely and cooperates with other ones through synchronizing actions. Also in PEPA model there is no need to implement Load Balancer directly – in can be realized implicitly through choice cooperator of the client component. Full PEPA model is listed below.

```
// Rate for intermediate actions
r=10000;
// Ratio of the requests to the WS_2.
d=0.7;
// CLIENT_0 – client, also reflects the presence of Load balancer
CLIENT_0=(move,r).CLIENT_0_0;
CLIENT_0_0=(move_2_0,d*r).CLIENT_0_0+
```

```

(move_1_3,(1-d)*r).CLIENT_0;
// Core2Duo Web Server 2 with two cores
WS_2_0=(move_2_0,T).WS_2_0_0;
WS_2_0_0=(move_1_1,r).WS_2_0
+(move_1_2,r).WS_2_0;
T_1 = (move_1_1,T).(comp_1, RateWS2).
(move_db, T).T_1;
T_2 = (move_1_2,T).(comp_2, RateWS2).
(move_db, T).T_2;
// Web Server 1
WS_1 = (move_1_3, T).(comp_3, RateWS1).
(move_db, T).WS_1;
// Database
DB_0=(move_db, RateDB).DB_0
CLIENT_0[40]<move_2_0,move_1_3> ((WS_2_0 <move_1_1, move_1_2> (T_1
<> T_2 ))<> WS_1) <move_db> DB_0

```

### 6.3 QPN Model

Elements of QPN formalism provide an easy way to represent architecture entities [9]:

- Web Server 1, Web Server 2, DB Server are mapped to queuing places with Processor-Sharing (PS) scheduling strategy. The service rates of places correspond to time demands of servers;
- Clients are mapped to queuing place with Infinite Server (IS) scheduling strategy which is used to represent clients sending requests to the system. The service time of this place corresponds to the average load;
- Thread pools of Web Servers are mapped to an ordinary place.

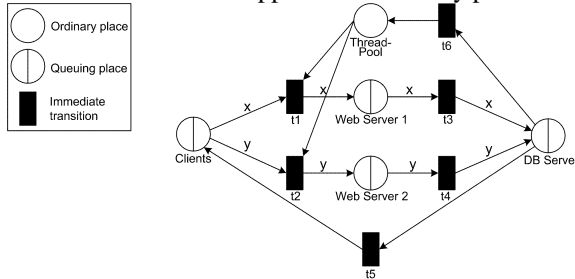


Figure 4. QPN Model of Test System

Two types of tokens (token colors) are used in the model for distinguishing requests between web servers: x and y. In QPN model Load Balancer was not implemented directly as it was done in PEPA. This implementation was realized implicitly through choice of immediate transition from a client request.

## 7. Models Calibration

Before starting models analysis we need to define  $Rate_{WS1}$ ,  $Rate_{WS2}$  and  $Rate_{DB}$  parameters in our models, i.e. calibrate our models. To accomplish it we performed a measurement of the transaction response time of the Web Server 2 with only one virtual user. Received response time of overall CPU load was about 0.6 second with average 35% percentage. Taken into account that this server has two cores and both of them equally participate in handling requests (due to used module `mod_fast_cgi` which starts several instances of PHP) the time to handle the request by one core is  $0.6 * 0.35 * 2 = 0.42$  sec. This is Web Server CPU depended part of the request processing and the estimation of the service rate of another CPU can be received by normalizing of this time to CPU frequency. For Web Server 1 we received:  $0.42 / 3.5 * 2.2 = 0.264$  sec. Among the Web Server CPU depended part of the request processing there is another part, which takes  $0.6 - 0.42 = 0.18$  sec. and it's treated as Database processing time. The rates of the models are defined as follows:

- $Rate_{WS1}: 0.42^{-1} = 2.38 \text{ sec}^{-1}$
- $Rate_{WS2}: 0.26^{-1} = 3.78 \text{ sec}^{-1}$
- $Rate_{DB}: 0.18^{-1} = 5.55 \text{ sec}^{-1}$ .

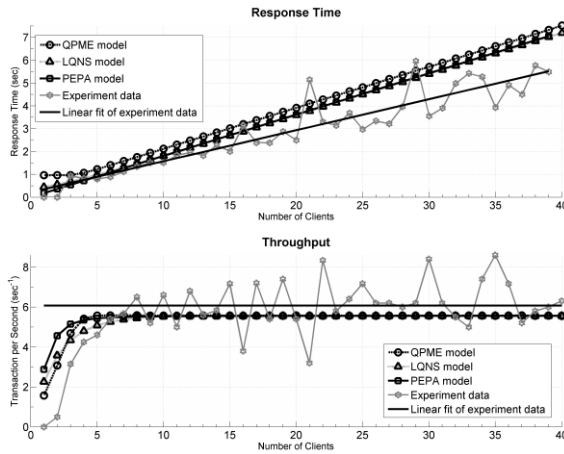
These values have been used in model analysis.

## 8. Models Analysis

During model analysis we made performance predictions of the system behavior under different external workload by varying number of clients simultaneously sending requests to the system from 1 to 40. Every model has been analyzed with its corresponding technique.

For LQN model we used `spex` utility, which is included in LQNS package. This tool reads parameterized LQN model, replaces parameter names with parameter values and then solves the model using `lqns` solver. `Spex` gathers measures of interest and creates text report with obtained data.

During analysis of PEPA model we used experimentation feature built-in in the PEPA Workbench. It provides the same functionality for PEPA models as `spex` for LQN models. Also has graphical user interface and is able to build graphs from gathered data.

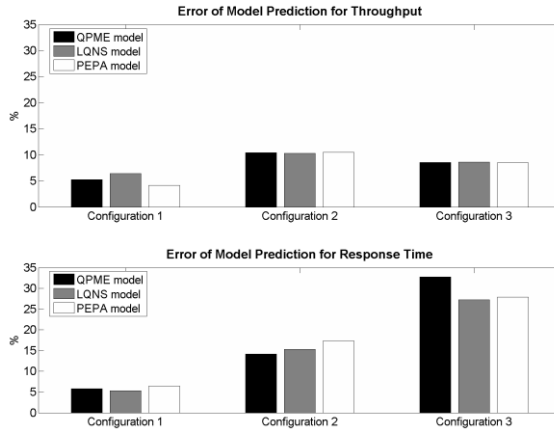


**Figure 5. Metrics for Configuration 3**

Analysis of QPN model has been done by SimQPN simulator of QPME package executed as a standalone Java application. QPME doesn't have any possibilities to simulate parameterized QPN models. That's why we developed a Perl script to simulate different workload by replacing parameters of QPN model in a loop and extract essential data from SimQPN output for further analysis.

Simulation results and experimental data for infrastructure 3 with Load Balancer are presented on Figure 5 and relative errors for all analyzed configurations are shown on Figure 6. It can be seen that our models are able to predict throughput (transaction per second) of the system with higher precision with errors less than 10%. While the predictions for response time are less precise and their errors are below 30%. The most possible reason of these errors is an influence of an unrevealed internal system structure. So it would be promising to analyze the effect of hidden queue buffers and request handle threads to model predictions.

It has been revealed that the fastest utility for model simulation is lqns. From another point of view QPME tool and PEPA Workbench provide convenient GUI editors for model creation.



**Figure 6. Relative Errors**

## 9. Conclusion and Future Work

During this study we have created performance models of the content management system “Joomla” using three different software performance description formalisms. Models have been calibrated using measurements of the test system with single-user workload. The comparison of the model simulation results with experimental data obtained by HP Load Runner has shown that all of the applied techniques are able to predict system behavior without detailed knowledge about the internal system structure. Difference from model predictions and experimental results lay in the acceptable area: for throughput it’s less then 10%, for response time – less then 30%. Such results make possible to use model predictions during early performance analysis of infrastructure for distributed business applications.

The investigation of errors caused by hidden structure of system components and methods to estimate them is the subject for further work.

## 10. References

- [1] Bause F., “Queuing Petri Nets – a formalism for the combined qualitative and quantitative analysis of systems”, 5<sup>th</sup> International Workshop on Petri Nets and Performance Models, Toulouse(France), pp 14-23, 1993.
- [2] Bause F., P. Buchholz, and P. Kemper. „Integrating Software and Hardware Performance Models Using Hierarchical Queuing Petri Nets”. In Proceedings of the 9. ITG / GI - Fachtagung Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen, (MMB’97), Freiberg (Germany), 1997.
- [3] Benjamin C. Lee, David M. Brooks, Bronis R. de Supinski, Martin Schulz, Karan Singh, Sally A. McKee, “Methods of Inference and Learning for

Performance Modeling of Parallel Applications”, PPOPP’07, San Jose, California, USA ,March 14–17, 2007.

[4] Franks G., S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside. “Performance analysis of distributed server systems”, 6<sup>th</sup> Int. Conf. on Software Quality (6ICSQ),pages 15-26, Ottawa, Canada, Oct. 1996.

[5] Gilmore S. and Tribastone M., “Evaluating the Scalability of a Web Service-Based Distributed e- Learning and Course Management System”, WS-FM 2006, LNCS 4184, pp. 214–226, 2006.

[6] Hillston J., “A Compositional Approach to Performance Modeling” Cambridge University Press, 1996.

[7] HP Load Runner, “HP essential knowledge series: an introduction to load testing for web applications”, White paper, 2007.

[8] Karlsson M., M.Covell, “Dynamic Black-Box Performance Model Estimation for Self-Tuning Regulators”, Proceedings of the Second International Conference on Autonomic Computing (ICAC’05)

[9] Kounev S., “Performance Engineering of Distributed Component-Based Systems – Benchmarking, Modeling and Performance Prediction”, Shaker Verlag, 2005.

[10] Omari T., Franks G., Woodside M., Pan A., “Solving Layered Queueing Networks of Large Client-Server Systems with Symmetric Replication”, WOSP’05, July 12-14, 2005

[11] Yin L., S.Uttamchandani, R.Katz, “An Empirical Exploration of Black-Box Performance Models for Storage System”, 14th IEEE International Symposium on Modeling, Analysis, and Simulation, 2006

[12] Apache HTTP Server, <http://httpd.apache.org/>

[13] CMS Joomla, <http://www.joomla.org/>

[14] Layered Queueing Network Solver software package, <http://www.sce.carleton.ca/rads/lqns/>

[15] PEPA Workbench, <http://www.dcs.ed.ac.uk/pepa/>

[16] PHP, <http://www.php.net/>

[17] Queueing Petri net Modeling Environment, [http://sdq.ipd.uka.de/people/samuel\\_kounev/projects/QPME](http://sdq.ipd.uka.de/people/samuel_kounev/projects/QPME)

# **Introduction to Continuous Integration or Stone Soup**

**Andrey Satarin**  
**Customized InformSystems**  
**www.custis.ru**  
**email: asatarin@custis.ru**

## **Abstract**

Continuous integration in software development process promises many advantages: fast defect discover, integration problems remove, fewer numbers of defects [1,2]. With more detailed consideration its turned out that this practice depends on others, such as unit testing, coding standard etc. Many of expected advantages are not realized without using such additional practices. Paradox situation occurs, where its unknown does continuous integration have independent value or all value comes from “foreign” methods. Isn’t where any fraud, when on the pretext of continuous integration adoption one tries to use other engineering practices advantages? Perhaps, continuous integration is “stone soup”, all ingredients are well-knows, but now served together with another title. In this article we try to show that this is not true, and continuous integration has its own value. This value considerably lower than synergetic effect from several practices, but adoption and use costs are much lower too. Adoption of “bare” continuous integration could also be the first step to many others effective development technologies.

**Keywords:** continuous integration; quality, agile development.

# **Введение в непрерывную интеграцию или каша из топора**

**Андрей Сатарин**  
**Заказные ИнформСистемы**  
**www.custis.ru**  
**email: asatarin@custis.ru**

## **Аннотация**

Использование непрерывной интеграции в процессе разработки программного обеспечения обещает много преимуществ: быстрое обнаружение ошибок, устранение проблем интеграции, меньшее число дефектов [1,2]. При более подробном рассмотрении, оказывается, что эта практика сильно зависит от других, таких как модульное тестирование, стандарт кодирования и т.д. Множество ожидаемых преимуществ не реализуются без использования этих дополнительных практик. Складывается парадоксальная ситуация, когда не ясно, имеет ли непрерывная интеграция независимую ценность или вся ценность обусловлена только «сторонними» методиками. Нет ли здесь обмана, когда под предлогом внедрения непрерывной интеграции пытаются использовать преимущества других инженерных практик? Возможно, непрерывная интеграция представляет собой «кашу из топора», все ингредиенты которой давно известны, но теперь поданы вместе под другим названием. В данной статье мы пытаемся показать, что это не так, и непрерывная интеграция имеет свою ценность. Эта ценность существенно ниже, чем синергетический эффект от нескольких практик, но и затраты на внедрение и использование существенно ниже. К тому же, внедрение «голой» непрерывной интеграции может служить и первым шагом к многим другим технологиям эффективной разработки.

**Ключевые слова:** непрерывная интеграция; качество, гибкая разработка.



## 1. Введение

Непрерывная интеграция (continuous integration, далее НИ) первоначально была создана как одна из практик экстремального программирования. Моментом создания считается приблизительно 2000 г., когда была написана первая версия статьи Мартина Фаулера [1]. С того времени она развивалась и изменялось понимание того, как она должна взаимодействовать с другими практиками гибкой разработки.

Образцом современного понимания НИ можно считать недавно вышедшую книгу Пола Дюваля [2]. В этом достаточно большом и подробном труде выделяется несколько составных частей НИ:

- НИ баз данных;
- непрерывное тестирование;
- непрерывную инспекцию кода;
- непрерывное развертывание;
- непрерывная обратная связь.

Все эти части подробно описаны и показано как они влияют на процесс разработки. В таком понимании практики НИ можно указать один недостаток — она не является самостоятельной и основная часть выгод получаемых от нее происходит не из нее самой. Выгоды появляются от удачного комбинирования данной практики с другими, такими как: модульное тестирование, автоматическое приемочное тестирование, автоматизированные инспекции кода (рис. 1). С одной стороны хорошо, когда много разнообразных практик вместе дают синергетический эффект, но с другой стороны, это поднимает входной порог использования этой технологии. Т.е. для использования НИ нужно уже иметь модульные тесты, стандарт кодирования и систему его автоматической проверки и т.д. Эта ситуация схожа с описанной в русской народной сказке «Каша из топора» [3]. Постулируется огромная польза от внедрения и использования НИ в проектах разработки ПО, но не уточняется, что большая часть этой пользы происходит не из самой практики, а из ее удачной комбинации с другими сильными методиками. На самом деле, описанные выше условия не являются необходимыми для использования НИ, можно получить выгоду от самостоятельного внедрения НИ,

с минимальным привлечением сторонних активностей. В данной статье обсуждается именно такой «упрощенный» способ.

## **2. Зачем это нужно?**

Может показаться что «урезанный» вариант НИ, не соответствующий последним достижениям в данной области, бесполезен и только будет отнимать время на внедрение и поддержку. Но это не так, есть круг проблем, которых можно просто и эффективно решить при помощи НИ, не отягощенной другими практиками. Также внедрение НИ, отдельно от других практик, может быть приемлемо для организаций, процесс разработки которых далек от гибких (agile) методологий или для организаций, которые только встают на этот путь. Проблемы, которые на наш взгляд, можно решить внедрением «голой» НИ, описаны в следующем разделе.

### **2.1. Проблемы**

В целом эти проблемы можно разделить на два вида, те, с которыми сталкиваются тестировщики и те, с которыми сталкиваются разработчики. Для других участников процесса разработки практика НИ также является полезной, но это не так явно выражено.

Любое тестирование начинается с получения и развертывания одной из версий ПО. Тестировщики, как правило, не могут собрать проект самостоятельно, для этого у них нет знаний, инструментальных средств (IDE), да и просто навыков. Но любое тестирование начинается с получения готовой к использованию бинарной сборки ПО и его развертывания на тестовом стенде. Для разработчиков, наоборот, собрать проект не составляет труда, эту операцию они проделывают часто и много. Кажется, это и есть решение проблемы. Нет. При

передаче версии от разработчиков могут возникнуть сомнения в целостности передаваемой сборки:

- Из какой версии исходных кодов она была собрана?
- Какие ключи компиляции и настройки среды окружения применялись?
- Не были ли внесены в код изменения, не отраженные в системе контроля версий?
- Были ли использованы правильные версии внешних библиотек при сборке?

На эти вопросы ответить не так просто, а проконтролировать каждого разработчика — еще сложнее. Многие скажут, что для сборки, развертывания и передачи ПО в тестирование должна существовать особая роль — инженер по сборке (build engineer), или даже целое подразделение сотрудников, но в небольших проектах такое просто невозможно. Конечно, эту роль может исполнять кто-то из членов команды, и, на первый взгляд, это кажется разумным. На самом деле, это лишь способ скрыть проблемы, но не избавиться от них. Возникнет консервация уникального знания и навыка в голове одного человека и связанные с этим риски:

- Что будет, если работник, ответственный за сборку и развертывание, уволится или заболеет?
- Какие есть гарантии, что он не сделает ошибки при очередной сборке или развертывании?

Можно идти путем регламентов, которые контролируют окружение и процедуру сборки, но такие регламенты сложно исполнять и еще более сложно поддерживать. Программисты не могут заниматься исполнением желаний тестировщиков: собирать проект по требованию, жестко контролировать окружение, в котором происходит сборка. Это тупиковый путь.

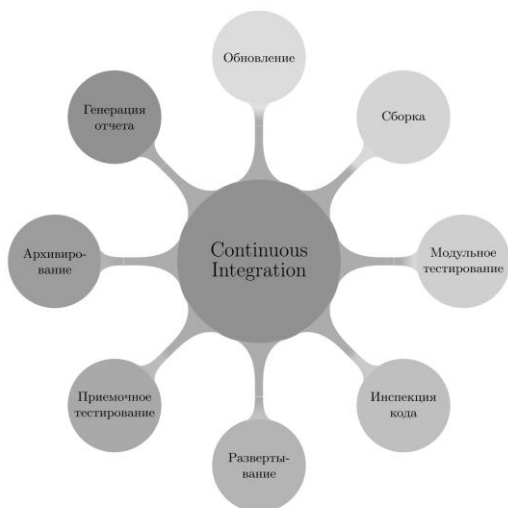


Рисунок 1. Составные части полного процесса непрерывной интеграции.

Сами разработчики тоже часто являются источником проблем. Такие проблемы могут быть охарактеризованы следующими фразами:

- «На моей машине все собирается и работает!»
- «На последней сборке, ушедшей в тестирование, используется устаревшая версия библиотеки superlib».
- «Я не могу работать, потому что проект не собирается. Я не знаю, когда все сломалось».

Особенно хороша первая проблема, вокруг этой фразы («It works on my machine!») даже возникла субкультура [4], но разработчики все равно продолжают повторять ее снова и снова. В борьбе с этим злом НИ особенно хороша.

Кому-то эти проблемы могут показаться смешными или надуманными, тем не менее, в разработке такое часто случается. Раннее обнаружение этих проблем экономит много времени и денег.

В итоге получается, что есть некий объем работ (сборка, развертывание и т.д.), который должен выполняться для того, чтобы контролировать (в некоторой степени) состоянии производимого ПО. Практика НИ призывает избавиться от этих рутинных операций посредством их автоматизации и передаче машине.

### 3. Решение

Автоматизация здесь — это в первую очередь автоматизация сборки, развертывания, обратной связи. Все эти автоматические процедуры связываются воедино на специальной интеграционной машине, а управление процедурами осуществляет сервер интеграции. Базовая схема организации процесса интеграции показана на рис. 2. На схеме выделены следующие этапы:

- Обновление исходного кода;
- Сборка;
- Развертывание проекта;
- Архивирование бинарных файлов;
- Генерация и публикация отчета.

Это практически минимальный набор этапов, при котором можно говорить о процессе НИ. В некоторых случаях можно убрать этап развертывания, если для программы этого не требуется. Остальные этапы, безусловно, необходимы. Рассмотрим подробно, что происходит на каждом из указанных этапов. Вначале сервер интеграции обновляет дерево исходных кодов проекта до последней свежей версии, это можно делать как по расписанию, так и при каждом изменении исходных кодов. Далее идет сборка. Сборка на интеграционном сервере отличается от сборки на локальной машине разработчика. Чем они отличаются и зачем? Современные сервера интеграции позволяют передавать метку (номер версии) в качестве параметра сборки. Рекомендуется включать этот параметр на видном месте в результирующий продукт, например, в информации о программе, которую может посмотреть пользователь. Кроме того, окружение на сборочной машине

контролируется более жестко. После этого, готовый продукт разворачивается в окружении. Важным моментом является, какое именно окружение использовать для развертывания в данном случае. Один из авторов практики НИ Мартин Фаулер советует делать это в промышленном окружении [2].

После развертывания идет этап архивирования, на котором происходит сохранение результатов сборки для дальнейшего использования.

Таким использованием может быть тестирование, демонстрация, воспроизведение проблем в ранних версиях. Кроме того на данном этапе может происходить пометка кода тегом в репозитории, для того, чтобы можно было всегда повторить данную сборку. Обычно тег совпадает с номером версии, ранее сгенерированным сборочным сервером.

После того, как все сделано, наступает этап генерации отчета и сервер оповещает разработчиков о результатах. Оповещение можно разделить на два типа: активное и пассивное. К первому относится email, системы мгновенных сообщений, мониторы. Второй тип — это публикация на веб или файловом сервере.

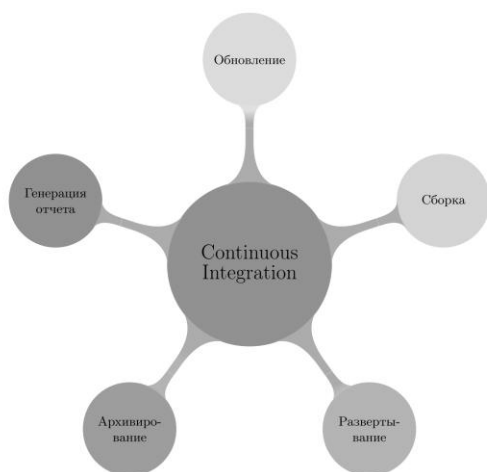


Рисунок 2. Составные части базового процесса непрерывной интеграции.

#### 4. Пример

Рассмотрим пример использования описанного подхода «голой» непрерывной интеграции в проекте разработки веб-приложения на платформе java.

В качестве средства управления версиями использовался Subversion, который поддерживается большинством широко распространенных серверов сборки. Сами средства сборки стандартны для платформы java — это ant и maven. Такая комбинация с одной стороны предоставляют большие возможности при управлении сборкой проекта в целом, а с другой стороны позволяет просто автоматизировать смежные этапы.

Более нестандартной является реализация этапа развертывания. В первую очередь из-за того, что стандартного механизма развертывания для различных серверов приложений не

существует. Мы, в данном случае, мы выбрали достаточно простой способ. Развертывание происходит на том же сервере, что и сборка, поэтому возможна подмена файлов приложения. После такой подмены сервер приложений (в нашем случае Resin) сам производит развертывание.

К сожалению это не все, что нужно сделать для запуска веб приложения, необходимо так же развернуть свежую версию базы данных. Свежую, значит ту, которая находится в репозитории, а где ее еще хранить? Для этой проблемы так же нет стандартного решения. Изначально мы пошли по очень простому и неправильному пути — стали хранить полный дамп базы в репозитории. Недостатки данного подхода очевидны: невозможность простого сравнения версий, неудобство работы с бинарным файлом. Гораздо более прогрессивным оказался метод хранения описания базы в SQL. Все современные средства работы с базой позволяют просто и удобно проводить экспорт таких скриптов. Такой подход к хранению структуры базы оказался существенно более удобным и не имеет указанных выше недостатков.

Последние этапы процесса непрерывной интеграции: архивирование и генерация отчета легко реализуются средствами практически любого специализированного сервера.

В итоге получилась система, которая производит сборку, развертывание и оповещение в случае каких-либо проблем. Конечно, она не способна отловить все проблемы (а какая система способна?), но даже такой простой «фильтр» умеет вылавливать некоторые из них. Прочие преимущества и недостатки описаны в следующем разделе.

## **5. Преимущества и недостатки подхода**

Выгода первая — избавление от рутины. Не надо объяснять, что профессионалы не любят рутину, кроме того, время людей становится все дороже, а время машин все дешевле.

Выгода вторая — простота и повторяемость. Кто угодно — любой участник проекта: тестировщик, аналитик и т.д. может



собрать и запустить проект, потому что все эти операции автоматизированы. Совершить ошибку при этом невозможно. Выгода третья — незаметность. На первых этапах использования большинству участников разработки нет необходимости что-то менять в их работе. Сравните это с такими «тяжелыми» практиками как разработка через тестирование (TDD), где разработчику надо практически поменять мировоззрение или внедрение автоматического приемочного тестирования, где тестировщику предлагается писать код. Если изменения в работе людей практически отсутствуют, они не будут сопротивляться внедрению и использованию НИ.

Как уже отмечалось, при комбинировании НИ с другими практиками получается синергетический эффект, но этот случай за пределами рассмотрения данной статьи.

Есть ли недостатки у практики НИ? Безусловно, но они не так значительны как преимущества. Например, к недостаткам относят необходимость иметь выделенный сервер, тратить время на поддержку работы этого сервера. Первый аргумент с каждым днем все слабее и слабее, железо стоит все меньше и меньше по сравнению с временем разработчика. Относительно второго аргумента можно сказать о нашем опыте. Время на поддержание сервера совершенно незначительно по сравнению со временем на обнаружение проблем другим способом и запоздалое их устранение.

## **6. Заключение**

Практика НИ тесно переплетена с другими практиками гибкой (agile) разработки: модульное тестирование, приемочное тестирование, стандарт кодирования, но она может применяться и без них. Мы не отрицаем преимуществ этих практик, особенно в комбинации с НИ, но их внедрение — отдельная задача. Внедрить несколько таких «ресурсоемких» технологий сразу невозможно, как бы ни призывали различные гуру. Это можно сделать только по частям. Например, сначала внедрить «голую» НИ, а затем к ней постепенно присоединять

другие методики. При этом необходимо последовательное «наращивание» шагов включаемых в сборку.

Т.о. непрерывная интеграция объединит вокруг себя несколько других инженерных практик, и благодаря этому может стать центром вашего процесса разработки.

## 7. Литература

[1] Мартин Фаулер (Martin Fowler) “Continuous Integration”  
<http://www.martinfowler.com/articles/continuousIntegration.html>

[2] Duvall, Paul and Matyas, Steve and Glover, Andrew  
“Continuous Integration. Improving Software Quality and Reducing Risk”, Addison-Wesley, 2007.

[3] «Каша из топора», русская народная сказка  
<http://www.skazki.org.ru/view.php?id=7073> см. так же  
[http://en.wikipedia.org/wiki/Stone\\_Soup](http://en.wikipedia.org/wiki/Stone_Soup)

[4] “It works on my machine!”  
<http://www.google.com/search?q=It+works+on+my+machine!>

# **Web-application testability evaluation**

**Alexei Barantsev**  
**Institute for System Programming**  
**of RAS**  
**email: barancev@ispras.ru**

## **Abstract**

This article introduces new aspects to the notion of web-application testability. Review of existing investigations related to software testability is provided. Restrictions of existing approaches and ways to overcome these restrictions are discussed. The article provides preview of a novel web-application testability evaluation tool.

**Keywords:** Software testing; software testability; software quality; software maintainability; web-applications.

# Оценка тестопригодности веб-приложений

Баранцев Алексей  
Институт системного  
программирования РАН  
email: barancev@ispras.ru

## Тезисы

Данная работа вводит новые аспекты в рассмотрение понятия тестопригодности веб-приложений. Приводится обзор работ, в которых исследуется понятие тестопригодности программ; обсуждаются недостатки существующих подходов, после чего предлагаются пути преодоления этих недостатков; приводится описание прообраза инструмента, реализующего предложенные идеи применительно к веб-приложениям.

**Keywords:** Тестирование программ; тестопригодность программ; качество программ; сопровождаемость программ; веб-приложения.

## 1. Введение. Обзор работ

Существующие работы можно разбить на две группы, в которых понятие тестопригодности рассматривается в одном из двух контекстов – реализационном либо спецификационном. Спецификационный контекст допускает более формальный подход к определению тестопригодности, поскольку это область, в которой можно работать с математическими моделями программ. Реализационный контекст ближе к практике, здесь исследователям приходится больше опираться на эмпирические знания, поэтому в этом контексте превалируют неформальные определения тестопригодности.

Обзор работ частично опирается на статьи Баумгартена [1] и Брунтинка и Дёрсена [3]. Баумгартен рассматривал исключительно вопросы тестопригодности спецификаций, оставляя в стороне реализационные аспекты. Брунтинк и Дёрсен, напротив, больше внимания уделяли практической стороне дела.

Первая попытка формально определить понятие тестопригодности принадлежит, видимо, Карнапу [4]. Разумеется, никакого отношения к тестированию программ

работы Карнапа не имели, он пытался определить, какие логические утверждения можно считать проверяемыми (testable), а какие нельзя. Тем не менее, современные исследования тестопригодности продолжают заложенную Карнапом традицию.

Кальман в работах по теории управления [7] сформулировал определение тестопригодности для детерминированных конечных автоматов. Он определил тестопригодность как комбинацию двух других свойств:

- управляемость (controllability) – возможность достигнуть любого состояния, подавая на вход различные стимулы,
- наблюдаемость (observability) – возможность определения состояния системы путём подачи на вход определённых стимулов и наблюдения полученных в ответ на эти стимулы реакций.

Чуть позже Штарк [9] расширил понятие наблюдаемости так, что оно стало применимо также для недетерминированных конечных автоматов.

Неформальные обсуждения понятия тестопригодности программ можно найти в [2], [5], [10]. В этих работах тестопригодность трактуется как наличие у программы тех или иных характеристик, облегчающих её тестирование. При этом «облегчение» может выражаться либо в сокращении стоимости, времени или усилий, либо в возможности применения тех или иных методов тестирования или инструментальных средств.

Принимая во внимание тот факт, что тестирование является инженерной, а не чисто теоретической дисциплиной, практические ограничения, такие как стоимость или время имеют не меньшее значение, чем теоретическая возможность создания набора тестов.

Поэтому усилия исследователей нацелены на то, чтобы для распространённых частных случаев определить некоторые совокупности характеристик программ и свести их к некоторому набору метрик, которые могут быть вычислены на основе реализации, и для которых имеется корреляция с полученными эмпирическим путём оценками затраченных на тестирование усилий.

Фридман [5], так же как и Кальман [7], определил понятие тестопригодности для программ как сочетание управляемости и наблюдаемости. Однако для управляемости и наблюдаемости он дал более широкие определения в терминах входных и выходных значений, поскольку программу не всегда можно трактовать как конечный автомат. Согласно Фридману,

- управляемость определяет, какую часть области выходных значений можно покрыть, подавая на вход все возможные варианты входных значений,

- наблюдаемость характеризуется тем, получаем ли мы на выходе разные или одинаковые значения на выходе, подавая на вход те или иные данные.

Если представить себе программу как функцию из области входных данных в область выходных данных, то можно сказать, что инъективная функция обладает хорошей наблюдаемостью, а сюръективная функция – хорошей управляемостью.

Воас и Миллер [10] также рассматривали отношение мощности множества выходных данных к мощности множества входных данных. Они изучали связь этого показателя с такой характеристикой, как степень чувствительности программы к наличию дефектов, то есть возможность обнаружения дефекта в программе, если известно, что он там имеется.

Другие исследователи пытались опираться не на «внешние», а не «внутренние» метрики, которые могут быть вычислены на базе исходного кода программы. Джунгмейр [6] изучал влияние связей между частями программы на сложность интеграционного тестирования. МакГрегор [8] аналогичное исследование проводил в отношении связей, характерных для программ, написанных на объектно-ориентированных языках – наследования, делегирования, исключений.

## **2. Тестопригодность веб-приложений**

Все изученные нами работы неявно предполагают два фактора, которые, вообще говоря, нельзя принимать за истину:

- тестопригодность зависит только от тестируемой программы,

- интерфейс взаимодействия тестовой системы с тестируемой программой не влияет на тестопригодность.

В данной главе мы рассматриваем эти два фактора, показываем, как можно их учесть, а в следующей главе описывается прообраз инструмента, предназначенного для оценки тестопригодности веб-приложений с учётом этих двух факторов.

### **2.1. Зависимость от инструментов, целей и методов тестирования**

Когда исследователи пытаются определить тестопригодность в терминах метрик, присущих самой

программе, они полагают, что тестопригодность можно рассматривать как некую функцию программы:

$$\text{Testability} = F(\text{Program})$$

Как было показано выше, исследователи пытаются свести эту функцию к вычислению определённых метрик, базирующихся либо на исходном коде программы, либо на области входных и выходных значений. Вряд ли у кого-либо вызовет сомнения тот факт, что сложность программы коррелирует с усилиями, которые необходимы для её разработки, тестирования и поддержания. Однако определение сложности программы не так-то просто свести к измерению тех или иных характеристик исходного кода. Поэтому такой способ относительно хорошо работает для простых случаев – модульное функциональное тестирование, и практически неприменим в других случаях.

Действительно, если посмотреть на типичное веб-приложение, можно видеть, что часть функциональности реализуется на стороне браузера на языке JavaScript или его аналоге, часть на сервере приложений, и часть – в базе данных на языках PL/SQL, TransactSQL или подобных им. Нам не удалось найти ни одной работы, в которой приводился бы способ оценки тестопригодности на основе анализа исходного кода для приложений, представляющих собой подобную совокупность разнородных частей.

Кроме того, все встреченные нами работы были посвящены оценке пригодности программы для функционального тестирования. Можно предположить, что анализ программного кода не позволяет оценить пригодность программы для других видов тестирования – производительности, удобства использования, защищенности и т.д.

Нам кажется, что эти исследователи в своих работах неявно предполагали и поэтому не упоминали про одну крайне важную вещь: тестирование всегда нацелено на достижение определённых целей. Эти цели могут быть различными, что и порождает разные виды или подвиды тестирования. Так, при тестировании производительности и функциональности ставятся разные цели. Аналогично, при модульном и системном тестировании цели также различаются. То есть тестопригодность является функцией не только тестируемой программы, но также цели тестирования:

$$\text{Testability} = F(\text{Program}, \text{Target})$$

Но и это ещё не всё. Чтобы понять, чего не хватает, проведём ещё одну аналогию, поговорим о достижимости некоторого места. Достижим ли Китай? Достижим ли необитаемый остров посреди Тихого океана? Достижимо ли

дно Марианской впадины? Достижима ли обратная сторона Луны? Для ответа на этот вопрос нужно знать, какими техническими средствами мы располагаем. Когда-то на любой из этих вопросов ответ был однозначный – нет, но по мере развития техники эти места стали достижимыми.

Точно так же тестопригодность зависит не только от самой программы, но также и от имеющихся в нашем распоряжении инструментальных средств:

$$\text{Testability} = F(\text{Program, Target, Tool})$$

Чуть ниже мы подробнее обсудим эту зависимость тестопригодности от инструментальных средств на примере веб-приложений. А пока отметим только, что в каждом конкретном случае нужно принимать во внимание не абстрактный «уровень технологического развития человечества», но конкретные инструменты, которые либо уже имеются, либо могут быть приобретены, не выходя за рамки доступного бюджета.

Следует также иметь в виду, что инструменты могут поддерживать несколько различных методов тестирования. Так, например, для веб-приложений возможно тестирование как на уровне браузера (то есть эмуляции действий пользователя с клавиатурой и мышью), так и на уровне HTTP-протокола. Поскольку для программы использование одного способа может оказаться проще, чем использование другого, а также инструменты могут лучше поддерживать тот или иной способ и хуже другие способы, мы явно подчеркнём эту зависимость от выбранного способа тестирования:

$$\text{Testability} = F(\text{Program, Target, Tool, Method})$$

Таким образом, можно сказать, что в большинстве имеющихся работ изучается частный случай этой функции, когда три последних параметра фиксированы, а именно: целью является модульное функциональное тестирование с оценкой полноты покрытия программного кода, используются традиционные инструменты модульного тестирования, способ тестирования – методом «чёрного» либо «белого» ящика на уровне программных интерфейсов (API).

Разумеется, мы не претендуем в данной работе полностью описать указанную функцию для всех возможных случаев. Мы всего лишь ставим целью показать, что имеется большой простор для развития исследований не только вглубь, но и вширь.

Вернёмся ещё раз к аналогии с понятием достижимости какого-либо места. Для повышения достижимости у нас могут быть два пути – либо совершенствование транспортных средств, либо улучшение инфраструктуры, которое позволит



использовать уже существующие транспортные средства. Так, чтобы обеспечить возможность доставки грузов в труднодоступное место, можно либо купить (а может быть даже изобрести) вертолёт, либо построить железную или автомобильную дорогу.

Точно так же можно говорить о повышении тестопригодности программ, причём тут тоже можно действовать с двух сторон – либо делать более хитроумные инструменты тестирования, либо разрабатывать приложения с учётом тестопригодности для уже имеющихся инструментов.

Опираясь на вышеприведённую формулу тестопригодности:

$$\text{Testability} = F(\text{Program, Target, Tool, Method})$$

можно видеть, что при этом мы фиксируем два параметра – цель и способ тестирования, и варьируем один из оставшихся параметров (или оба сразу).

## **2.2. Тестопригодность интерфейса**

Второй серьёзный недостаток имеющихся работ проистекает из того, что в них рассматривается тестирование на уровне программных интерфейсов. При таком предположении можно считать, что мы можем подавать на вход любые допустимые спецификацией интерфейса стимулы, а также можем без труда наблюдать реакции тестируемой программы.

Однако на практике это предположение часто не выполняется. Например, при тестировании программ с графическим пользовательским интерфейсом, как подача стимулов, так и наблюдение реакций представляют собой отдельную технически весьма сложную задачу. При тестировании компонентов, которые должны работать в определённом окружении, например, сервлетов или EJB, зачастую используют искусственное окружение (псевдо-контейнер).

Поэтому, в дополнение к общему понятию тестопригодности программы, можно ввести более узкое понятие тестопригодности того или иного интерфейса, имея в виду управляемость – возможность и простота отправки стимулов через этот интерфейс, и наблюдаемость – возможность и простота наблюдения реакций.

Для веб-приложений широко используется два способа функционального тестирования – на уровне HTTP-запросов и на уровне браузера.

Первый способ хорош тем, что тестовая система получается независима от браузера и чаще всего даже от операционной системы. Но в современных веб-приложениях зачастую часть логики реализуется в виде скриптов на языке JavaScript, которые исполняются в браузере. Поэтому такой способ применим только к программам, в которых вся логика реализована на серверной стороне.

Второй способ требует достаточно тесной интеграции инструментов тестирования с различными браузерами и способности инструментов эмулировать различные действия пользователя.

С точки зрения простоты отправки стимулов и наблюдения реакций интерфейс HTTP-запросов обладает лучшими показателями, формирование и анализ HTTP-запросов и ответов на них является технически более простой задачей, чем эмуляция действий пользователя в браузере – нажатий кнопки мыши и её движений, нажатия клавиш на клавиатуре.

Однако с точки зрения возможности отправки всех возможных стимулов и наблюдения всех возможных реакций при наличии скриптов, исполняемых в браузере, первый способ явно проигрывает второму, несмотря на увеличивающуюся при этом техническую сложность.

Для повышения тестопригодности интерфейса, как уже отмечалось выше, можно предпринимать усилия с двух сторон – либо проектировать приложение так, чтобы инструментам было проще получить доступ к элементам интерфейса, либо создавать инструменты, способные работать с произвольными интерфейсами. Покажем это на конкретном примере.

В литературе можно найти советы по проектированию веб-интерфейсов, нацеленные на повышение тестопригодности, среди которых чаще всего встречается рекомендация использовать статическое именование структурных элементов, то есть указание значений атрибутов `id` или `name`, которые заранее определены, а не генерируются динамически и не меняются впоследствии при модификации приложения. Однако в последнее время можно заметить, что инструменты стали более мощными, использование для идентификации элементов интерфейса языка запросов XPath позволило практически полностью отказаться от статического именования этих элементов.

### 3. Прообраз реализации

В настоящее время в Институте системного программирования РАН разрабатывается инструмент для оценки тестопригодности веб-приложений, который учитывает два рассмотренных выше фактора.

Инструмент предназначен для того, чтобы на ранних стадиях проекта по созданию веб-приложения, когда уже готов статический дизайн страниц, но приложение ещё находится в разработке, помочь выбрать наиболее подходящий для тестирования инструмент и способ тестирования, либо выдать рекомендации по доработке интерфейса веб-приложения, так чтобы можно было воспользоваться имеющимися инструментами тестирования.

Для поддержки работы этого инструмента создана информационная система, в которой описываются поддерживаемые различными инструментами цели и способы тестирования, а также анализируются и фиксируются ограничения инструментов тестирования, такие как:

- возможность работы на уровне HTTP-протокола,
- интеграция с различными браузерами,
- возможность поддержки сессий,
- возможность интерпретации JavaScript,
- возможность работы с диалоговыми окнами,
- поддержка различных способов аутентификации, и т.д.

Инструмент анализирует структуру страниц веб-приложения и определяет, насколько хорошо тот или иной инструмент сможет работать с этим интерфейсом, после чего сообщает, какие проблемы могут возникнуть при использовании тех или иных инструментов.

Это даёт возможность более осознанно подбирать способы и инструменты тестирования, что положительно сказывается на качестве программы и сокращает расходы на разработку.

### 4. Список литературы

[1] B. Baumgarten, H. Wiland. Qualitative Notions of Testability. IWTCS 1998: 345-360

[2] B.Beizer. Software Testing Techniques. 2<sup>nd</sup> ed. V. Nostrand Reinhold, 1990

- [3] M. Bruntink, A. van Deursten. Predicting Class Testability using Object-Oriented Metrics. 4th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'04)
- [4] R. Carnap. Testability and Meaning. *Philosophy of Science*, 3/4, pp 420-471, 4/1, pp1-40, 1936/37
- [5] R.S. Fridman. Testability of Software Components. *IEEE Transactions on Software Engineering*, 17/6, pp553-564, 1991
- [6] S. Jungmayr. Identifying test-critical dependencies. In *Proceedings of the International Conference on Software Maintenance*, pages 404–413. IEEE Computer Society, October 2002
- [7] R.E. Kalman, P.L. Falb, M.A. Arbib. *Topics in Mathematical System Theory*. McGraw-Hill, 1969
- [8] J. McGregor, S. Srinivas. A measure of testing effort. In *Proceedings of the Conference on Object-Oriented Technologies*, pages 129–142. USENIX Association, June 1996
- [9] P.H. Starke. *Abstrakte Automaten*. Deutscher Verlag der Wissenschaften, 1969
- [10] J.M. Voas, K.W. Miller. *Software Testability: The New Verification*. IEEE Software, May 1995

# **Project Audits: Agile Teams Self-Assessment**

**Dmitry Lobasev**  
**Agile coach**  
**email: dlobasev@luxoft.com**

## **Abstract**

On a certain level of maturity every Agile team asks itself – are we doing Agile or not? Do we comply, for example, combination of Scrum & XP? When your company has five, ten or twenty such teams it is necessary to track and correct them in order to provide an appropriate company-wide level of process and product quality.

Since there is no global Agile certification in the world, many teams created their own Agile checklists – team assessment forms to appreciate their agility. What to do - find and use one of existed forms or create your own – it depends on your process specific and your experience in teams assessments. For example, Luxoft started to use a self-assessment form from Dean Leffingwell – an author of Scaling Software Agility book. During last three years, while we make Agile teams assessments, we created our own self-assessment form based on external known forms and our expertise, especially in distributed Agile.

Our experience in team's assessments shows us that it is very useful both for a company in general and for each team under assessment. For a company it creates a big picture of what is going on in Agile projects. For a team it makes a good input data for retrospectives and helps to warn about potential problems that are not visible inside a team.

**Keywords:** What is Agile, Teams assessment, Retrospectives, Audits.

# Проектные Аудиты: Система Самооценки Agile Команда

Дмитрий Лобасев  
Тренер и консультант по  
гибким методологиям  
email: dlobasev@luxoft.com

## Тезисы

Рано или поздно у каждой команды возникает вопрос - насколько тот процесс, по которому она работает, соответствует классическому Agile подходу, например часто встречающейся комбинации методологий Scrum + XP. Когда же в вашей компании таких команд не одна, а пять, десять или двадцать, то уже появляется необходимость в том, чтобы так или иначе отслеживать и корректировать процесс разработки в каждой команде, добиваясь некой "стандартизации", и, соответственно, определенного уровня качества.

Отсутствие глобальной Agile сертификации привело к тому, что существует достаточно большой набор различных "чеклистов" - опросников (team assessment forms), которые призваны оценивать "гибкость" той или иной команды. Какую из существующих форм выбрать или создать свою – зависит от специфики вашего процесса и, конечно, опыта в оценке команд. Так, например, компания Люксофт начала проводить оценки своих команд по опроснику, предложенному Дино Леффингвелом в его книге *Scaling Software Agility*. За три года, которые в компании существует практика оценки команд, мы создали свой собственный опросник, на основе опыта как сторонних команд, так и специфики распределенного Agile процесса Люксофт.

Как показывает опыт, наличие процесса оценки Agile команд полезно как в целом для компании, позволяя увидеть картину по всем проектам, так и для отдельных команд, предоставляя хорошие входные данные для проведения ретроспектив и позволяя предупредить возникновение многих проблем, симптомы которых очень часто не видны внутри команды.

**Keywords:** Что такое Agile, Оценка Agile команд, Ретроспективы, Аудиты.

## **1. Введение**

На текущий момент не существует единой глобальной Agile сертификации. Видимо потому, что Agile подход настолько «гибок», что к нему нельзя применить какие-то определенные стандарты. Точно также, как нельзя однозначно сказать, посмотрев на проект и команду, "Agile" она или нет, потому что каждый новый проект под каждого нового заказчика будет чем-то отличаться от предыдущих. Однако, существует достаточно большой набор различных «чеклистов», как правило составленных практикующими Agile экспертами, имеющими большой опыт работы с различными командами и заказчиками.

## **2. Аудит или Оценка?**

Мы привыкли, что чаще всего, когда речь заходит о проверке чего-либо на соответствие чему-либо, называть это аудитом. А человека, который аудит проводит, аудитором. И звучит все это настолько формально, что основные цели проверки обычно теряются: команда во главе с менеджером проекта старается подделать факты под соответствие проверкам, например, создавая макеты необходимых артефактов. И команда воспринимает аудит как просто лишние заботы, которые лично ей (команде) совсем не нужны - дайте просто спокойно работать.

Но ведь изначально цель аудита была посмотреть, как идут дела в проекте, и при необходимости, на основе опыта аудитора, дать грамотный совет, который был бы воспринят командой и способствовал бы повышению эффективности ее работы.

Поэтому, оставляя аудиты для СММІ и прочих формальных моделей, для Agile команд, которые по сути своей максимально неформальные, более разумно использовать термин Оценка. И позиционировать оценку следует в качестве помощи команде обнаружить неявные проблемы, дать входные данные для ретроспектив.

Но все-таки основное отличие оценки от обычного аудита заключается в том, что не аудитор, а сами члены команды, выражая свое личное и независимое мнение по тому или иному вопросу, формируют результаты оценки, которые потом обобщаются и показывают очень реалистичную картину проекта.

### 3. Цели проведение оценки

Здесь можно выделить две группы целей, цели компании и цели команды.

Для компании процесс оценки позволяет:

- Иметь полное и главное реальное представление о том, что происходит в проектах компании;
- Отслеживать и поддерживать определенный уровень качества проектов, и, как результат, выпускаемых продуктов;
- Покрывать СММІ практику независимой проверки качества процесса - Process and Product Quality Assurance (PPQA).

Для команды же, оценка позволяет:

- Предупредить возникновение части проблем, потому что в большинстве случаев симптомы не видны внутри самой команды; их можно обнаружить только при внешнем анализе;
- Повысить «зрелость» команды за счет применения новых неиспользуемых практик, которые содержатся в форме опроса. Это опять же, позволяет держать все проекты компании на качественно одном уровне;
- Иметь входной набор данных для ретроспектив – очень часто члены команд указывают комментарии к вопросам, более подробно высказывая свое мнение.

### 4. Что оценивается

Оценка обычно осуществляется по двум направлениям: соответствие процесса разработки в команде принципам гибкой разработки (Agile Manifesto) и наличие практик одной или нескольких гибких методологий. Если в первом случае проверяется, например, вовлеченность заказчика в процесс разработки (участие в митингах, постоянный контакт с разработчиками), то во втором случае это фиксированность длины итерации или написание модульных тестов на весь новый код.

Ранее упоминалось, что не существует единых параметров оценки Agile проектов. Каждая компания выбирает подходящие под свой процесс критерии, на основании которых и проводится оценка. Таким образом, например, команда Люксофт, набравшая 70 баллов, может быть ничуть не «более



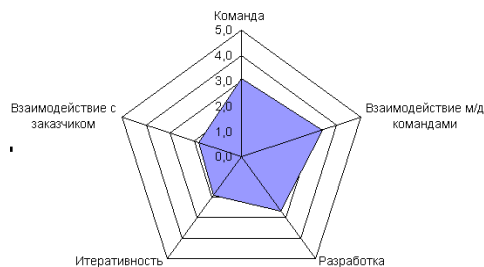
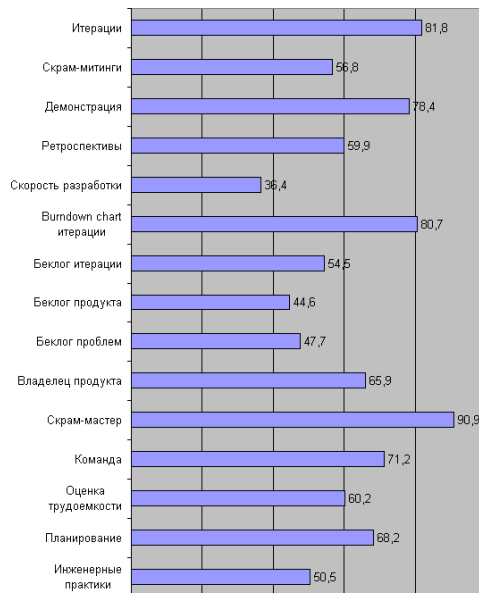
гибкой», чем команда IBM, набравшая 50 баллов. Потому что используются приблизительно одинаковые, но в то же время разные критерии оценки.

## **5. Формат проведение оценки**

В общем виде проведение оценки команды выглядит следующим образом. Оценка проводится на регулярной основе, начиная от одного раза в две итерации (для новых команд) до одного раза в релиз (для более опытных). Важно напомнить команде, что опрос в принципе анонимен, т.е. соответствие фамилия-анкета не хранится нигде, кроме почты человека, который проводит опрос.

Всем членам команды рассылается опросник, в котором на каждый вопрос можно выразить свое мнение (от 0 до 5 или от Никогда до Всегда), а также указать комментарий, уточняющий ответ или более полно его раскрывающий. После того, как вся команда заполнила анкеты, формируется сводный отчет, который показывает общий балл команды (от 0 до 100) и усредненные баллы по группам вопросов (например, Планирование или Инженерные практики).

Формат представления результатов может быть различным, но чаще всего используются следующие варианты, в зависимости от уровня группировки вопросов:



Хорошей практикой является публикация внутри компании таблоида – рейтинга проектов, который содержит названия проектов и их общие баллы. Поскольку процесс оценки команд непрерывный, то положение того или иного проекта в рейтинге часто меняется, создавая некий дух соревнований. Более того, за первое место по результатам месяца, можно давать команде ценный приз, например, в виде книги от Agile guru.

## 6. Заключение

Как показывает опыт, наличие системы оценки Agile команд позволяет иметь полную и реалистичную картину с состоянием проектов в целом по компании, а так же детальные данные от членов команды отдельно взятых проектов, что

позволяет оценить не только проект в целом, но и выявить тревожные моменты, касающиеся отдельно взятых членов команды.

Безусловно, система оценки должна эволюционировать со временем и обязательно учитывать комментарии членов оцениваемых команд, которые очень часто бывают достаточно важными.

Но в тоже время, никакая система оценки не гарантирует вам успеха вашего проекта, даже если команда набрала 100 баллов из 100. Это всего лишь цифры. Однако, детальный анализ результатов оценки, от общего (команда) к частному (человек), будет хорошим помощником для улучшения процесса разработки вашего проекта и повышения эффективности работы команды.

## **7. Ссылки**

[1] Ken Schwaber. “Agile Project Management with Scrum”, Microsoft Press. 2004.

[2] Dean Leffingwell. “Scaling Software Agility”. Addison Wesley, 2007.

[3] Mary and Tom Poppendieck, “Implementing Lean Software Development”, Addison Wesley, 2007.

[4] Esther Derby and Diana Larsen. “Agile Retrospectives”, Pragmatic Bookshelf. 2006.

# **Segmentation of IT customers on internal market**

**Fedor Krasnov**  
**MBA student at MIRBIS,**  
**Moscow, Russia**  
**email: FKrasnov@akado.ru**

**Alexander Sergeev**  
**Professor EMBA program at**  
**MIRBIS Business School,**  
**Moscow, Russia.**  
**email: A.Sergeev@mirbis.ru**

## **Abstract**

The increased attention given to information technology in the business environment has propelled information systems (IS) and IT Departments into prominent positions in many firms. Yet, little is known about how IT Departments exercise communications. The value of ISs is losing in IT Departments, which plays considerable role in the value chain from vendor to corporate customers. Poor communication effectiveness of internal IT departments has a bad influence on how IS will introduced and exploit. This paper addresses that gap through its focus on one research question: Who are the customers of the services IT Departments provide? In the paper, we use Electronic Document System consumption data as our model building samples and present a modeling framework for building segment-level predictive models that utilize pattern based clustering approach and signature discovery techniques. We devise fluctuate-rate matrix to study various modes. Through clustering on this matrix, we uncover different customer characteristics. Utilizing these characteristics, we can build two-dimension Consumption- Based customer segmentation model.

**Keywords:** Organizational engineering, IT management.

## **Функция или услуга?**

### **(Внутренний рынок услуг в организации.)**

Чем занимается каждое подразделение в компании? Выполняют возложенные на него функции, говорит классический менеджмент. Но понятие функции плохо связано с такими смыслообразующими для бизнеса идеями как результат и эффективность. Действительно, что значит хорошо выполнить свою функцию? У функции нет заказчика и потребителя. Поэтому начиная с 80-х годов прошлого столетия функции, выполняемые определенными подразделениями в организации, начали рассматриваться управленческой наукой как услуги, предоставляемые внутри организации. Это могут быть услуги по проектированию, закупкам, бухгалтерскому учету и т.п. С приходом персональных компьютеров появился новый вид внутрикорпоративных услуг ИТ, потребителями которых стали сотрудники почти всех подразделений компании.

Такой взгляд на процессы, происходящее внутри организации, побудил авторов данной статьи к новому взгляду на отношения между поставщиками и потребителями внутренних услуг. Ключевым для этого анализа становится понятие - «рынок внутренних услуг». Ведь если это рынок, то для того чтобы на нем успешно работать следует его понять и охарактеризовать. В этом контексте возникает ряд последовательных вопросов. Например, к какому типу массовый (B2C) или корпоративный (B2B) относится этот рынок? Какова специфика коммуникаций между потребителями и поставщиками на этом рынке? Есть ли конкуренция на этом рынке?

#### **ИТЛ и реальная жизнь**

Рассмотрим более подробно эти вопросы применительно к ИТ-подразделению и к услугам, которые оно оказывает внутри организации. Исторически рассмотрение деятельности ИТ-подразделения, как сервисного, предоставляющего услуги, развивалось следующим образом:

- ❑ в 1986 году Агентство ССТА (Великобритания) расценило затраты правительства на ИТ как высокие (около £8 млрд/год) и инициировало работы по изучению опыта в области IT Service Management;
- ❑ в августе 1989 года, родилось название ITIL (IT Information Library), как области знаний, предназначенной для консолидации и последующей стандартизации все информации по данному вопросу в библиотеки книг;
- ❑ в 1992 году начата публикация первой версии библиотеки. В разработке сильно помогли голландцы 90-е, середина ITIL распространяется за пределы правительственных организаций и за пределы Великобритании;
- ❑ к 1998 году опубликована последняя книга первой версии (всего 34 книги). Инициирован проект по обновлению библиотеки;
- ❑ в сентябре 2000-ого опубликована первая книга второй версии. А в ноябре 2004, опубликована последняя книга второй версии (всего 8 книг).

Таким образом, к 2004 году были формализованы процессы предоставления ИТ-услуг. Но внедрение методики ITIL до сих пор остается сложной задачей. По мнению авторов, основной проблемой при внедрении ITIL является отсутствие в ней понятия «клиента». Услуги ITIL полностью игнорируют все, что было разработано в маркетинге, начиная с 60-х годов прошлого века, включая такие базовые понятия как продукт-микс (4P), brand, и.т.п.

Не удивительно, что такое узко-технологическое рассмотрение проблемы создания сервисного подразделения стимулировало авторитарный подход ИТ-руководителей, так как, рыночные механизмы формирования услуг не принимались во внимание при внедрении ITIL. В итоге ИТ-

услуги внедрялись только как монополия и нуждались в регуляторных мерах со стороны генерального директора. Как только эти меры ослабевали ИТ, как услуги, переставали существовать.

### Управление требованиями

С другой стороны в течение последних 7 лет по данным аналитического агентства Gartner бюджеты на внедрение новых ИТ систем, Информационную безопасность и ИТ-инфраструктуру растут из года в год на 7-10% процентов. Из этого можно сделать два важных вывода:

- обоснование инвестиций в ИТ становится все более сложным. Генеральному Директору становится все труднее понять затраты на ИТ-проекты и эксплуатацию ИТ-систем;
- растет количество ИТ-систем в организации, а значит, растет и количество ИТ-услуг по предоставлению доступа к функциям ИТ-систем для различных подразделений.

В этой связи в организации растет значение Директора по ИТ в организации. Именно он общается с Генеральным Директором, Юристами, Финансовым Директором и помогает решить что «Дорого», а что «Дешево» что эффективно, а что нет при решении проблем компании или каждого из ее подразделений. Априори не понятно, например, ИТ-проект стоимостью 200 тыс. долларов и продолжительностью 6 месяцев – это дорогой проект или дешевый? А если без этого проекта нельзя сделать отчетность по МСФО и компания не сможет пройти международный аудит? Кто решает, адекватны ли сроки, цена и задачи, решаемые для такого проекта?

Решение подобных вопросов требует интенсивного взаимодействия ИТ- и генерального директоров. Однако, в подобных дискуссиях ИТ-Директорами как правило исповедуется подход, называемый в дисциплинах по разработке программного обеспечения «управление требованиями». Для Директоров по ИТ такой подход в отношении к заказчику представляется естественным обобщением их опыта разработчиков программного обеспечения. Однако, практика показывает, что такой подход быстро заводит в тупик все

переговоры о выделении бюджета. Причина в том, что для постановки выполнимых требований к ИТ-системе необходимо понимать ее возможности. На практике же оказывается, что требования исходят из представлений о гипотетических возможностях ИТ-системы и поэтому ставят ИТ-Директора в тупик.

Практика использования подхода управления требованиями пользователей, применяемая в разработке программного обеспечения, при решении ключевых вопросов развития ИТ в рамках организации оказывается тупиковой. И понятно почему. Наивное стремление ИТ-Директора удовлетворить все требования своих клиентов, потребителей ни как не учитывало опыт хорошо проработанного в маркетинге понятия клиентоориентированности. С точки зрения маркетинга клиентоориентированность – это умение создать ожидания у выделенной части клиентов и превзойти их. Из этого определения следует два важных вывода:

- ИТ-Директору нужно управлять не требованиями, а ожиданиями своих потребителей;
- только превышая ожидания (по срокам, затратам, и.т.п.) можно показать дополнительные результаты и оправдать дополнительные затраты, которые неизбежны в любом ИТ проекте. В отличии от ожиданий требования нельзя превысить, им можно только соответствовать или не соответствовать.

#### Клиентоориентация в коммуникациях

Рассмотрим более подробно коммуникацию между ИТ-Директором и Генеральным Директором с точки зрения развития организации. Очевидно, что Генеральный Директор частично делегирует полномочия по принятию решений ИТ-Директору. Частичность делегируемых ИТ-Директору полномочий обуславливается его сферой компетенций, то есть ИТ. Но, ИТ-Директор определяет не сам для себя развитие ИТ в организации, качество ИТ, состав ИТ-услуг, и.т.п. ИТ должны



соответствовать развитию других видов деятельности организации и, в первую очередь коммерческой, которая нередко сильно зависит от возможностей ИТ. Получается замкнутый круг. В этом замкнутом круге кто-то должен взять на себя роль первым предложить необходимое решение.

Разорвать этот замкнутый круг можно если снова опереться на понятие клиентоориентации. Если ИТ-директор – продавец, а генеральный директор – его клиент, то именно ИТ-директору целесообразно предложить необходимое решение. . Именно ИТ-Директор должен создать те ожидания от ИТ, которые он сможет выполнить. Из этого следует, что ИТ-Директор должен предложить ИТ-решения Генеральному Директору. Такое предложение ведет к дополнительным затратам и рассматривается Генеральным Директором как предложение совершить покупку, а Директор по ИТ рассматривается как продавец ИТ-решений. Этот вывод приводит нас к тому, что рассмотрение взаимодействия ИТ-Директора и Генерального Директора будет эффективным только, если ИТ-Директор сумеет использовать накопленный в маркетинге опыт и теоретические знания, будет подготовленным продавцом.

Развивая применения маркетинговых концепций к коммуникациям ИТ-Директора внутри организации, нужно отметить, что огромное значение в нем имеют факторы времени и языка. Генеральный Директор не даст более 5 -10 минут на объяснения и не будет вникать в технические термины. Поэтому необходимо подготовить и отрепетировать коммуникационную идею, которую вы хотите донести.

Далее в Таблице 1 приведены два примера коммуникационных идей простого технического продукта из книги Дж. Гриндера.

Модель А.	Модель Б..
Поздравляю Вас, Вы только что купили великолепный фонарик «Эверлайт». Ничего нет проще Вашего фонарика «Эверлайт» - Вам надо только нажать на серебряную полосу со стороны стекла. Готово! Теперь Вам светло даже в абсолютной темноте. Не забудьте, что перед включением надо вложить в это устройство две батареи.	Этот механизм состоит из простой цепи, соединяющей источник тока (две батареи «С», включенные последовательно) с лампой накаливания, помещенной в конце механизма. Простая скользящая рукоятка, продвинутая к концу, где находится лампа, замыкает цепь и включает лампу, создавая источник освещения.

Таблица 1. Коммуникационная идея фонарика.

Из примеров в Таблице 1 очевидно, что «Модель Б» не годится для коммуникации с Генеральным Директором и вообще с не специалистами, однако именно к такому типу объяснений склонны прибегать ИТ-специалисты. Стоит ли удивляться тому, что Генеральный директор не хочет покупать такую ИТ-услугу.

Второй аспект -- можно придумать идеальную концепцию позиционирования, но не суметь ее донести. Поэтому, несомненно, важным является вопрос о каналах и способах коммуникации. Надо ответить на следующие вопросы: это будет служебная записка или презентация проекта, она будет выражена в словесной или визуальной форме? Чем в большей степени ИТ-директор сумет понять, как будет «удобней» воспринимать предполагаемую информацию его «клиенту», чем с большей результативностью он справится с поставленной задачей.

Серьезное отношение к результатам любого проекта требует системного подхода к коммуникациям. Мировой опыт и такие гуру как С.Питерс, доказали, что наиболее продуктивным является отношение к проекту как к бренду. Следуя предложенной идеологии на проект должны

распространяться такие важные свойства бренда как узнаваемость и лояльность потребителей. Мы для решения этой задачи используем хорошо зарекомендовавшую себя модель лидерства, Leadership Equity Models (LEM). По сути, это классификация коммуникационных стратегий, основанная на анализе ведущих мировых и российских брендов.

Согласно LEM, существует четыре модели построения модели лидерства в проектах, которые условно называются Power, Identity, Community и Explorer. Все они обладают определенными признаками и требуют той или иной последовательности действий в коммуникации с потребителем. Много ИТ-проектов бывают не успешными из-за того, что ИТ-директора не учитывают модель лидерства, на которую ориентируется их клиент – генеральный директор. В таблице 2 показаны модели лидерства применительно к ИТ проектам, которые характерны для топ-менеджеров компаний. В каждой из них разное позиционирование результатов проект и коммуникационная стратегия. Доказано, что коммуникационная стратегия выигрышная в одной ситуации может привести к краху в другой.

	<b>Power</b>	<b>Identity</b>	<b>Explorer</b>	<b>Community</b>
<b>Целевая аудитория</b>	Ценители наибольшей пользы	Ориентированные на эмоции	Новаторы	Универсальная
<b>Позиционирование результатов проекта</b>	Лучший результат	Самовыражение, узнай себя	Достигни большего	Осознавать себя частью большего
<b>Как коммуницировать?</b>	Лучшее решение конкретной проблемы	Использование эмоциональных ценностей потребителя	Ориентация на новые возможности, результат всегда с плюсом	Создание истории, культивирование символов и атрибутов
<b>Примеры традиционного и ИТ рынка</b>	Tide – всегда белоснежное бельё  1С Бухгалтерия – программа для реальных директоров	Мегафон – Будущее зависит от тебя  MS Dynamics	Intel – воспитание маленьких Эйнштейнов  SAP R/3	Страна Marlboro  Open Source ERP (Millennium BSA, Compiere)

Таблица 2. Модели лидерства (Leadership Equity Models, LEM) применительно к ИТ проектам.

\*\*\*

В заключение необходимо отметить, что маркетинг ИТ-проектов и услуг становится все более востребованным в практике ИТ-Директоров. Есть хорошие примеры ИТ-проектов ставших брендами в компаниях, о которых будет рассказано в следующих статьях. Опыт авторов от общения с ИТ-Директорами таких проектов показывает, что они чаще всего интуитивно пришли к такому положению вещей, но очень довольны тем, что получилось.

# **Software Transactional Memory system for C++**

**Serge Preis**

**serguei.v.preis@intel.com**

**Segey Kozhukhov**

**sergey.s.kozhukhov@intel.com**

**Aleksei Cherkasov**

**aleksei.g.cherkasov@intel.com**

**Tian Xinmin**

**xinmin.tian@intel.com**

**Ravi Narayanaswamy**

**ravi.narayanaswamy@intel.com**

**Ali-Reza Adl-Tabatabai**

**ali-reza.adl-tabatabai@intel.com**

**James Cownie**

**Intel Corporation**

**james.h.cownie@intel.com**

**Robert Geva**

**robert.geva@intel.com**

**Intel Corporation**

## **Abstract**

Multicore hardware recently received wide acceptance in all areas of computing from servers to notebooks and now it requires novel software to unleash full power of parallelism. By this reason technologies for parallel programming especially for shared memory systems (such as multicore) are getting significant boost nowadays.

Among hot topics of ongoing researches in this area are easy to use and scalable concurrency control mechanisms and one of those is Transactional Memory (TM). This paper presents pure software implementation of transactional memory system (STM) which

supports the following TM ideology of shared data accesses: all data modifications belonging to the same transaction (atomic section) become visible to other threads at the same moment or discarded altogether if conflict with other thread (contention) is detected, in this case transaction is restarted.

The system consists of C/C++ compiler supporting TM-specific language extensions and TM run-time library. This paper primarily focuses on language constructs and aspects of C++ support: the system presented is, perhaps, the first STM system which consistently supports C++ classes, inheritance, virtual methods, functional and class templates and exception handling

Transactional memory provides simple means for concurrency control and adds such advantages over traditional approaches as composability of transactional codes and failure atomicity. The cost of convenience is performance: TM and especially STM adds overhead on code in transactions. Some optimization helping to relax this problem are listed and discussed. Paper also shows comparative performance results of STM versus different lock techniques.

The main result of the presented work is publicly available full-featured STM system for C/C++ which accelerates development of concurrent systems providing syntax and semantics of single global lock while achieving scalability of fine-grain locking systems and gives some nice additional benefits.

**Keywords:** Software Transactional Memory; Concurrency control; Multithreading; C++ programming

# **Система Программной Транзакционной Памяти для C++**

**Сергей Прейс**  
**serguei.v.preis@intel.com**

**Сергей Кожухов**  
**sergey.s.kozhukhov@intel.com**

**Алексей Черкасов**  
**aleksei.g.cherkasov@intel.com**

**Шинмин Тиан**  
**xinmin.tian@intel.com**

**Рави Нараянасвами**  
**ravi.narayanaswamy@intel.com**

**Али-Реза Адл-Табатабаи**  
**ali-reza.adl-tabatabai@intel.com**

**Джеймс Коунн**  
**james.h.cownie@intel.com**

**Роберт Гива**  
**robert.geva@intel.com**

**Intel Corporation**

## **Тезисы**

Многоядерные аппаратные платформы, получившие распространение в последнее время требуют нового программного обеспечения, чтобы раскрыть весь заложенный в них потенциал. В связи с этим технологии параллельного программирования для систем с общей памятью (каковыми являются многоядерные системы) получили последнее время заметное развитие.

В частности, большое внимание уделяется исследованиям в области простых в использовании и эффективных механизмов

параллельного доступа к общим данным, одним из которых является Транзакционная Память. В данной статье пойдёт речь о чисто программной реализации системы транзакционной памяти. Транзакционная Память обеспечивает такой доступ к разделяемым данным, при котором изменения данных в рамках транзакции либо становятся видимы другим потокам все сразу, либо (в случае конфликта с другими транзакциями) отменяются и транзакция перезапускается.

Система состоит из компилятора для языков C, C++, расширенных конструкциями ТП, и библиотеки времени исполнения. В статье основной упор сделан на языковые расширения и поддержку C++ – представленная система ТП едва ли не первая для этого языка поддерживающая классы C++, их наследование, виртуальные функции, шаблоны и обработку исключений.

Транзакционная память - удобное средство управления доступом к разделяемым данным, предоставляющее ряд преимуществ: возможность переиспользования транзакционного кода в рамках другого транзакционного кода, атомарность обработки ошибок и т.п. Платой за удобство служит производительность – использование ТП вносит накладные расходы и потому оптимизация производительности – важный аспект представленной работы.

Основным результатом является общедоступная, полнофункциональная система программной транзакционной памяти. Её использование позволит ускорить построение эффективных параллельных систем за счёт сочетания простой программной модели аналогичной неименованным критическим секциям с масштабируемостью, достижимой лишь при использовании индивидуальных блокировок для данных.

**Keywords:** Программная транзакционная память; многопоточное программирование; синхронизация доступа; Программирование на C++



## 1. Введение

Распространение многоядерных платформ поставило перед разработчиками ПО новые задачи: чтобы полностью использовать мощность современных вычислительных систем необходимо разрабатывать параллельные алгоритмы, что заметно усложняет и алгоритмическую и архитектурную часть программных систем. В связи с этим средства упрощения разработки параллельных (многопоточных) приложений приобрели особую актуальность, и интерес к ним растёт год от года.

Одной из самых сложных задач в реализации параллельных алгоритмов с общей памятью (таково большинство многопоточных алгоритмов) является задача эффективной реализации доступа к разделяемым ресурсам. С одной стороны необходимо обеспечить корректное функционирование – отсутствие взаимных блокировок (deadlocks) и условий гонок (race conditions), с другой стороны простои из-за ожидания доступа не должны снижать масштабируемость производительности системы с ростом числа потоков/ядер.

Традиционным методом организации доступа к разделяемым данным являются различные виды блокировок (критические секции, семафоры и т.п.). Однако реализация эффективной синхронизации этими средствами может быть достаточно сложна, а отладка и поиск ошибок – ещё сложнее. Наиболее простой стратегией организации синхронизации является использование одной блокировки для всех разделяемых ресурсов, однако в большинстве случаев эта стратегия отрицательно влияет на масштабируемость производительности. Лучшие результаты даёт использование различных блокировок для различных ресурсов, но трудоёмкость этого подхода несравнимо выше.

Транзакционная память (ТМ – Transactional Memory) – одно из современных перспективных направлений исследований в области организации управления доступом к разделяемой памяти. Суть подхода в том, что все изменения данных изолированы в пределах транзакции и становятся видны либо все сразу на выходе из неё, либо отменяются и вся транзакция перезапускается сначала, если обнаружен конфликт с другой транзакцией.

Синтаксически работа с транзакционной памятью аналогична неименованным критическим секциям, однако, никакой из потоков, одновременно выполняющих транзакцию, не блокируется – все они выполняются параллельно. Двумя дополнительными преимуществами транзакционной памяти являются (1) возможность вложенных транзакций – код с транзакциями может быть переиспользован в рамках другого транзакционного кода; (2) возможность откатить транзакцию явно позволяет вернуть память в корректное состояние при обнаружении ошибки в транзакции.

По причинам, описанным выше, интерес к ТМ последнее время заметно вырос. Реализация транзакционной памяти требует того или иного мониторинга доступов к памяти, потому большинство работ относятся к управляемым (managed) языкам и системам программирования, таким как Java[9], C#[11], Haskell[10] и Caml[15]. Определённые исследования были посвящены C [17,6] и практически нет работ посвящённых C++ – отдельные аспекты рассматриваются в [1,5], но эти работы не претендуют на полноту. Безусловно, C и C++ остаются одними из самых распространённых языков в областях системного и высокопроизводительного программирования, потому наша работа достаточно актуальна.

Представленная в работе система – это чисто программная (без специальной аппаратной поддержки) реализация транзакционной памяти (STM – Software Transactional Memory) для C и C++. В отличие от предыдущих работ она

1. Поддерживает не только C, но и C++ включая классы, наследование, виртуальные функции, неявные конструкторы и деструкторы, шаблоны, управление памятью и обработку исключений, а также вызовы библиотечных функций, в том числе для ввода-вывода.
2. Основана на существующем высококлассном оптимизирующем компиляторе, предоставляет языковые расширения и набор оптимизаций для транзакционной памяти.
3. Включает высокопроизводительную динамическую библиотеку, поддерживающую несколько различных и переключаемых во время исполнения режимов работы, в том числе сериализованный режим для исполнения библиотечного кода. Интерфейс библиотеки позволяет

использовать различные алгоритмы STM и переключать их во время исполнения.

4. Представляет собой законченную и производительную реализацию STM, доступную для тестирования, экспериментов и предварительного использования. Результаты внутреннего тестирования производительности и масштабируемости на наборе тестов SPLASH2 представлены ниже в секции 5.

## 2. Расширения C++ для ТП

Представленная система расширяет C++ новыми языковыми конструкциями (а не прагмами компилятора, как, например, в [46, 28]) и, в отличие от ранних работ ориентированных только на C [17], поддерживает многие ключевые конструкции и механизмы C++.

### 2.1. Атомарные блоки

Основной конструкцией для транзакционной памяти, определяющей собственно область транзакционного исполнения, является атомарный блок (atomic block):

Пример 1:

```
__tm_atomic {  
    if (local_var > Global_val) {  
        Global_max = local_var;  
        local_var = Global_arr[++loc_i];  
    }  
}
```

Аналогично конструкциям **atomic** из ранних работ [1, 18] он определяет, что каждый такой блок исполняется как транзакция: атомарно и изолированно от других таких блоков.

Семантически атомарный блок эквивалентен неименованной критической секции: каждый атомарный блок изолирован и как бы целиком выполняется до или после атомарных блоков в других потоках. В действительности же все они выполняются параллельно, но без эффектов в других потоках. Внутренность атомарного блока трансформируется компилятором таким образом, что все доступы к глобальным данным обрабатываются вызовами STM-библиотеки. Это позволяет обнаруживать конфликты – одновременные доступы к одинаковым данным из разных транзакций, причём, как минимум из одной – по записи. В случае конфликта библиотека восстанавливает локально-изменённые данные к состоянию на

начало блока и, с новыми значениями глобальных данных, блок перезапускается снова до тех пор, пока не пройдёт без конфликтов (для предотвращения бесконечных откатов применяются специальные меры).

Любая передача исполнения за пределы атомарного блока вызывает попытку успешного завершения транзакции и публикации изменений. Это касается как обычного выхода из блока, так и выполнения конструкций `break`, `return`, `goto` и т.п.

Атомарный блок может содержать любые конструкции языка C++, допустимые в блоке кода, включая, другие атомарные блоки.

Пример 2:

```
__tm_atomic {  
    // Some code  
    __tm_atomic {  
        // some more code  
    }  
}
```

Все изменения сделанные во всём стеке вложенных транзакций станут видимы только при завершении самой внешней транзакции (closed nesting [28]). Однако, откаты, в том числе по запросу от пользователя (см. 2.2.) делается только до границы самой вложенной транзакции, так что границы вложенных транзакций не совсем прозрачны.

Вызовы функций внутри атомарного блока обрабатываются специальным образом: вызовы функции, имеющих транзакционную версию (порождённую компилятором, описанную пользователем или известную библиотечную), заменяются вызовами этой версии. Для косвенных вызовов вставляются динамические проверки наличия транзакционной версии. Прекомпилированные библиотечные функции, функции без транзакционной версии и функции ввода вывода вызывают переход транзакции в сериализованный режим: откат таких функций невозможен и потому для их выполнения должно гарантироваться отсутствие конфликтов, что и достигается сериализацией – глобальной блокировкой всех остальных транзакций.

## **2.2. Явные (пользовательские) откаты**

STM предоставляет расширенный сервис по сравнению с обычными критическими секциями: конструкция **`__tm_abort`**; откатывает исполнение текущей самой вложенной транзакции,

отменяет все сделанные в ней изменения и передаёт управление непосредственно за конец атомарного блока этой транзакции.

Конструкция `__tm_abort` может быть использована лексически только в атомарном блоке, она не может быть использована в функции вызванной из блока. Это ограничение вполне разумно, поскольку код, содержащий атомарный блок с явным откатом, должен быть написан в предположении, что тело атомарного блока может не исполниться.

Кроме того, откат по требованию пользователя невозможен, если до него случился переход транзакции в сериализованный режим. В этом режиме откаты невозможны, потому попытка отката вызовет ошибку времени исполнения, поскольку обнаружить эту проблему при компиляции не всегда возможно.

Однако во вложенной транзакции такой откат допустим.

#### Пример 3:

```
__tm_atomic {  
    cout << "HelloWorld!"; //precompiled  
    __tm_atomic {  
        // some code  
        __tm_abort; // OK, inner TX aborted  
    }  
    if (some_condition) {  
        __tm_abort; // runtime error  
    }  
}
```

Случаи, когда откат будет гарантированно вызван в транзакции после прекомпилированной функции обнаруживаются компилятором и диагностируются как ошибка, остальные случаи совместного использования в атомарном блоке прекомпилированного кода и явного отката вызывают предупреждение компилятора.

### **2.3. Разметка функций**

Чтобы транзакция не сериализовалась при вызове функции, код функции должен быть обработан компилятором примерно так же, как внутренность атомарного блока. Но поскольку ровно эти же функции могут быть вызваны не из транзакции, то они должны существовать в двух видах – транзакционном (клонированном) и обычном. Для обеспечения отдельной компиляции функций и транзакций, их вызывающих, а также для более точного указания свойств функций, введена

возможность разметки функций для транзакционной компиляции.

Идея разметки функций не нова, однако описываемая система предоставляет несколько больше вариантов разметки, чем предыдущие работы [17], давая возможность более гибкого контроля над размером кода и наличием транзакционных версий.

Разметка задаётся с использованием ключевого слова `__declspec()` на Windows™ и `__attribute__()` на Linux. Примеры даны в варианте для Windows™. x

#### Пример 4:

`__declspec(tm_callable) int foo(int);`

Допускается разметка любых функций, в том числе методов классов и шаблонных функций. Далее приводятся основные пометки для функций. Полный набор пометок описан в [13].

**tm\_callable** – функция, которая может вызываться как из транзакции, так и из обычного кода. Функция полностью дублируется и одна из копий дополнительно обрабатывается компилятором для транзакционного исполнения и переименовывается. Специальные усилия предпринимаются для организации косвенных вызовов транзакционной версии.

Никаких ограничений на содержимое функций, помеченных таким образом, не накладывается, в частности можно вызывать в коде прекомпилированные функции – это вызовет сериализацию текущей транзакции.

Тело такой функции должен компилировать STM-компилятор и её реализация должна быть помечена также как и декларация. В противном случае редактор связей (linker) не найдёт переименованный клон функции, не сможет собрать программу и выдаст ошибку.

**tm\_pure** – функция, которую программист хочет выполнять в транзакции «как есть». Такая функция не должна обращаться к глобальной памяти и иметь побочных эффектов (либо глобальная память константная, а побочные эффекты не важны для исполнения программы). Такая функция, равно как и её вызовы, никак не обрабатываются компилятором, и потому такая пометка может стоять даже на декларации библиотечной функции. Ставя такую пометку, важно понимать, что функция может быть перевызвана многократно в процессе одного исполнения транзакции из-за возможных откатов.

К сожалению, в случае косвенного вызова библиотечная функция, помеченная таким образом, трактуется как прекомпилированная и потому вызывает сериализацию транзакции. Исключение составляют виртуальные `tm_pure` методы (как прекомпилированные, так и доступные в исходном коде), а также все `tm_pure` функции с соответствующей пометкой на реализации доступной STM-компилятору.

**tm\_safe** – аналогична `tm_callable`, но предполагает более строгие проверки: в теле функции запрещены вызовы функций, не помеченных явно как `tm_pure` или `tm_safe`, а также любые косвенные вызовы кроме виртуальных методов `tm_pure` или `tm_safe`.

`tm_safe` гарантирует возможность отката и перезапуска, что делает их вызовы безусловно совместимыми с `__tm_abort` в рамках одной транзакции.

## 2.4. Разметка классов

Разметка класса – это умолчание для разметки всех новых методов класса, включая порождаемые автоматически и виртуальные. То есть разметка класса применяется к методу, если на методе явно не указано другое.

Пример 5:

```
__declspec(tm_callable) class A {  
    __declspec(tm_safe) int getValue(); //safe  
    virtual int process(); //tm_callable  
}; // A() - implicit, tm_callable
```

Методы, унаследованные из класса предка, сохраняют свою разметку. Невиртуальные методы, переопределяющие методы предка, считаются новыми в классе, и на них действует разметка класса. В тоже время виртуальные методы, переопределяющие методы предка, считаются наследниками (см. 2.5). Это согласуется с правилами наследования/переопределения методов в C++.

Только `tm_callable`, `tm_safe` или **tm\_unknown** может быть указано для класса. Отсутствие пометки, как и на функции, означает отсутствие предположений о природе кода.

Для выборочной отмены пометки на методах `tm_callable/tm_safe`-класса введена ещё одна пометка для методов – **tm\_unknown**, кроме того, все перечисленные в 2.2. пометки функций, применимы и к методам классов.

## 2.5. Наследование разметки

Класс-наследник наследует и пометку класса предка. Для отмены пометки `tm_callable` служит пометка **`tm_unknown`** указанная на классе. Поскольку пометки на классах влияют только на методы принадлежащее непосредственно этому классу, то допускается произвольная смена разметки в иерархии (в отличие от наследования для виртуальных методов).

В случае множественного наследования для класса выбирается самая сильная разметка из предков. Сила возрастает в ряду `tm_unknown` (нет пометки), `tm_callable`, `tm_safe`.

Для наследования виртуальных функций действуют более строгие правила. Обработка вызовов виртуальных методов делается по статической информации типа указателя на объект, а он может быть типом предка, поэтому потомок должен иметь не менее сильную пометку, чем предок. В противном случае к вызову могут быть применены более строгие требования, которым потомок не отвечает.

В случае если при переопределении в классе-потомке пометка не указана, она наследуется от метода-предка.

Допустимые сочетания пометок предков и потомков виртуальных функций при переопределении сведены в таблице 1.

В случае множественного наследования из всех возможных пометок для метода выбирается самая сильная (`tm_pure` сильнее, чем `tm_unknown` и противоречит остальным). Если пометки предков противоречивы (напр., `tm_pure` и `tm_callable`), то выдаётся сообщение об ошибке. Выбранная пометка используется как умолчание при переопределении функции в потомке и для проверки совместимости переопределения по таблице 1, приведённой ниже.

Пример 6: (`#define` – для сокращения строк)

```
#define _ds(x) __declspec(x)
struct A {
    virtual int _ds(tm_safe) getA() const;
    virtual int process(); //unknown
};
_ds(tm_callable) struct B {
    virtual int getA() const; //tm_callable
    virtual int __ds(tm_unknown) process();
};
```



```

struct AB : A, B { //tm_callable
    int _ds(tm_pure) getA(); //tm_safe->error
    int process(); //tm_unknown
};

```

**Таблица 1. Правила переопределения**

Base	Derived class			
	unknown	callable	safe	pure
unknown	yes	Yes	yes	yes
callable	no	Yes	yes	no
safe	no	No	yes	no
pure	no	No	no	yes

## 2.6. Разметка шаблонов

К шаблонным функциям, методам и классам применимы в полной мере правила разметки для обычных функций, методов и классов соответственно. Кроме того, допускается переопределение разметки базового шаблона на специализации как полной, так и частичной, а также на явной инстанцииции шаблонов.

Пример 7:

```

template <class T> _ds(tm_pure) T max(T,T);
template <> char* //specialization
    _ds(tm_safe) char* max<char*>(char*,char*)
template int* //instantiation
    _ds(tm_callable) max<int*>(int*,int*);

```

## 2.7. Обработка исключений

Исключения, не пересекающие границу атомарного блока, обрабатываются по обычным правилам C++ даже если они лежат в атомарном блоке или клонированной версии функции. При этом обработчики, принадлежащие транзакции, обрабатываются компилятором и исполняются транзакционно, как и любой другой код. В частности это означает, что возможна ситуация, когда в обработчике исключения будет обнаружен конфликт и вся транзакция, включая исключение, будет отменена. В этом случае после рестарта исключение может либо возникнуть снова, либо нет. Окончательный результат транзакции будет включать только последнее (бесконфликтное) исполнение, а значит и факт наличия исключения будет определяться только этим последним исполнением.

Если же исключение покидает атомарный блок, то, как и любая передача управления за пределы блока, оно вызывает успешное завершение транзакции и публикацию изменений. Основания для такого поведения следующие:

1. Откат отменит изменения данных в рамках транзакции, и факт наличия исключения будет противоречить данным, анализ причины исключения будет затруднён либо невозможен.
2. Невозможно откатить состояние транзакции, если она к этому моменту исполнила прекомпилированный код и была сериализована. Попытка обработать исключение откатом привела бы к ошибке времени исполнения (см. 2.2).
3. Это полностью согласуется с поведением неименованных критических секций – их состояние не откатывается при исключениях.
4. Существует возможность перехватить исключение и сделать откат транзакции явно с помощью `__tm_abort`. Явный откат сделает поток управления более очевидным и позволит подготовить слепок состояния транзакции на момент возникновения исключения.

Следует, однако, заметить, что в очередной версии нашей системы будет экспериментальная возможность выполнения отката для исключений пересекающих границы транзакции.

## **2.8. Выделение памяти в транзакции**

STM-библиотека предоставляет специализированные реализации функций выделения и освобождения памяти для использования внутри транзакции. Это позволяет не сериализовать транзакции с выделением или освобождением памяти внутри и избежать при этом утечек или повторного освобождения памяти. Эта возможность особенно важна для C++, где выделение памяти может быть неразрывно связано с инициализацией объектов. В C предпочтительно выделять память за пределами транзакции, даже если инициализация этой памяти должна делаться внутри. Однако и для C система предоставляет транзакционные версии функций работы с памятью.

## **3. Оптимизирующий компилятор**

Большую часть работы по реализации семантики транзакционной памяти делает STM-библиотека, однако роль

компилятора в системе сложно переоценить – он не только предоставляет языковые средства, упрощающие работу с ТМ, но и играет заметную роль в полной и эффективной её поддержке во время исполнения.

Компилятор в нашей системе реализует самостоятельно или с помощью библиотеки следующие функции:

- Поддержку языковых расширений описанных выше. Именно ясные и простые языковые конструкции играют решающую роль в упрощении реализации доступа к разделяемым данным.
- Поддержку различных режимов работы ТМ, в том числе требующих дублирования кода функций и отдельных атомарных блоков.
- Расстановку вызовов к STM-библиотеке на границах транзакций и при обращениях к памяти
- Поддержку вызовов функций из транзакции включая:
  - Поддержку прямых вызовов клонированных версий для `tm_callable` функций;
  - Поддержку косвенных вызовов с динамической проверкой на наличие клонированной версии;
  - Поддержку вызовов `tm_pure` функций, в том числе и виртуальных методов;
  - Поддержку вызовов прекомпилированных функций с сериализацией транзакций;
- Поддержку обработки исключений;
- Ряд оптимизаций и прочих техник повышения эффективности исполнения, о которых будет сказано ниже.

Чтобы дать библиотеке больше контроля над реализацией STM наша система, в отличие от предыдущих работ [17], использует фиксированный интерфейс с библиотекой без включения элементов реализации непосредственно в порождаемый код. Таким образом, компилятор не может оптимизировать непосредственно код, реализующий алгоритм STM, но за то библиотека может менять стратегии динамически в процессе исполнения или может быть полностью заменена без перекомпиляции приложения. При этом компилятор всё же имеет существенный контроль за тем, как будет исполняться транзакционный код, каковы будут накладные расходы на его исполнение, какие классические оптимизации удастся

выполнить до ТМ-трансформаций. Библиотека предоставляет расширенный набор функций для поддержки оптимизаций.

Чтобы сделать исполнение транзакционного кода наиболее эффективным компилятор реализует следующие стратегии и оптимизации:

1. Вызовы к STM-библиотеке вставляются достаточно поздно в процессе компиляции, чтобы перед этим отработало максимальное число классических оптимизаций.
2. При поддержке библиотеки для повторных обращений к памяти используются специальные оптимизированные функции. Обращения по чтению могут вообще не заменяться вызовами, однако это сужает выбор возможных алгоритмов ТМ.
3. По транзакции вычисляются и передаются в библиотеку важные свойства, влияющие на выбор алгоритма ТМ.
4. Обращения к локальной памяти обнаруживаются и не передаются библиотеке, либо передаётся лишь значение для отката.
5. Короткие транзакции могут оставаться «как есть» и исполняться в режиме блокировки для снижения накладных расходов.
6. Известные библиотечные функции, распознанные компилятором, могут получать признак `tm_pure`.
7. Межпроцедурный анализ позволяет пометить функцию как `tm_callable` и даже `tm_pure` по её коду и использовать это при её вызовах.

Разработка оптимизаций ещё не завершена – возможности для дальнейшего повышения производительности далеко не исчерпаны.

#### **4. Немного о ТМ-библиотеке**

Специально для описываемой системы была разработана STM-библиотека, эффективно реализующая самые современные методики ТМ. Как было сказано выше, библиотека реализует фиксированный, но достаточно широкий интерфейс, позволяющий реализовать различные алгоритмы и переключать их динамически во время исполнения программы, при этом оставляя компилятору простор для оптимизаций.

Интерфейс содержит примерно следующий набор функций:

```
TxnDesc* getTransaction()  
int beginTx(TxnDesc*,int modes_and_hints)
```

```

void commitTx(TxnDesc*)
int beginInnerTx(TxnDesc*,int)
void commitInnerTx(TxnDesc*)
void abortTx(TxnDesc*)
void switchToSerialMode(TxnDesc*)
void write<Type>(TxnDesc*,Type*,Type)
Type read<Type>(TxnDesc*,Type*)
void logValue<Type>(TxnDesc*,Type*)
void logMem(TxnDesc*,void*,size t)
void writeAfterRead(Type*,Type)
void writeAfterWrite<Type>(Type*,Type)
Type readAfterRead<Type>(Type*)
Type readAfterWrite<Type>(Type*)
Type readForWrite<Type>(Type*)

```

Назначение большинства функций достаточно понятно. Функции вида *\*After\** и *\*ForWrite* используются для оптимизации повторных обращений к памяти.

На данный момент библиотека реализует 4 основных режима управления ТМ:

1. С оптимистическими проверками чтений
2. С пессимистическими проверками чтений
3. Приоритетный
4. Сериализованный

Первые два - это классическая транзакционная память с непосредственной блокированной записью и, соответственно, отложенной и непосредственной проверкой чтений. Третий режим позволяет отдать (долгой и конфликтующей) транзакции приоритет во всех конфликтах и таким образом исполнить её до завершения без откатов. Последний режим – глобальная блокировка и серийное исполнение транзакций.

В основном транзакции исполняются в одном из первых двух режимов (в каком конкретно – выбирается динамически), остальные используются для поддержки специальных случаев.

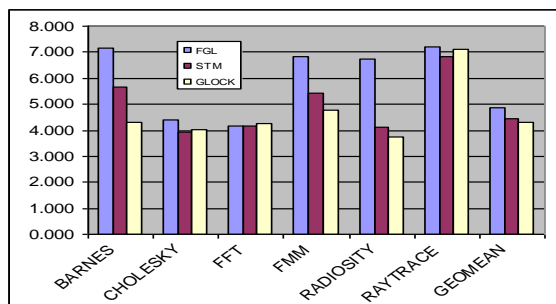
Режим исполнения для транзакции библиотека в основном выбирает сама на основе предыстории исполнения и характеристик, переданных компилятором, однако определённые режимы могут быть затребованы явно для исполнения ввода-вывода или оптимизаций.

Более подробно библиотека описана в [14]

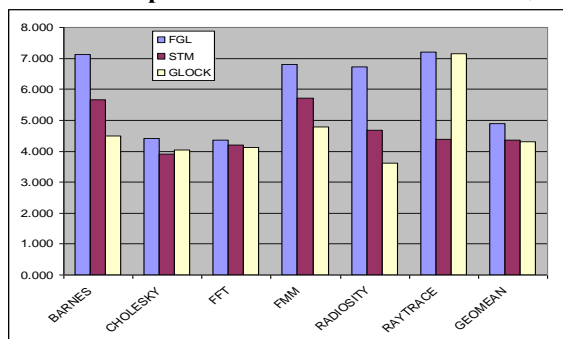
## 5. Производительность

Тестирование системы производилось на большом числе различных программ, включая SPLASH2 [18] и STAMP [2].

Ниже приводится ускорение исполнения SPLASH2 относительно



**Рис. 1. Ускорение на 8 потоках с оптимизацией**



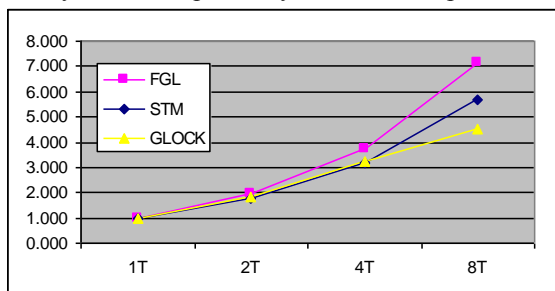
**Рис. 2. Ускорение на 8 потоках без оптимизации**

однопоточного исполнения для кода с собственными аккуратными блокировками (FGL), одной общей блокировкой (GLOCK) и нашей системой (STM) на 8и-ядерной машине.

Эффект от TM не одинаков для разных приложений. Так в RAYTRACE STM проигрывает даже общей блокировке – все транзакции там короткие и компилятор сериализует их, без этой оптимизации проигрыш заметно больше (см. рис.1). Однако заметно, что накладные расходы на запуск сериализованной транзакции всё же выше, чем при простой блокировке. Этот эффект особенно сильно проявляется когда используются все ядра в системе, так на 16и-ядерной системе для 8и потоков картина будет не столь драматичной.

RADIOSITY наоборот страдает от оптимизаций – транзакции в нём разнородные и безо всяких оптимизаций STM показывает лучший результат.

Эффект от использования ТМ проявляется только при достаточно большом количестве потоков – в нашем случае лишь начиная с 8 потоков на многих приложениях STM начинает ощутимо выигрывать у общей блокировки.



**Рис. 3. Ускорение с увеличением числа потоков**

## 6. Заключение

В течение ближайшего месяца общедоступная версия описанного компилятора будет обновлена. Она будет основана на новейшей версии базового компилятора и будет включать исправление ошибок предыдущих версий, поддержку возможностей описанных в статье и даже несколько больше. Но работа продолжается – введение новых оптимизаций позволит ещё поднять производительность кода и устранить многие слабые места, кроме того, ведутся работы по устранению ненужного клонирования и сокращению размера бинарного кода.

С другой стороны всё больше серьёзных игроков на рынке ведут работы в области транзакционной памяти: интерес к ТМ высказывают Sun Microsystems [5,7], IBM [12], HP [2], AMD[8] и другие. Технология постепенно выходит за рамки чисто научных исследований – пока состояние разработок ещё не позволяет применять ТМ повсеместно, но постепенно она получает признание: в частности она используется для доступа к общим данным в таких суперкомпьютерных языках как Fortress от Sun[16], X10 от IBM[4] и Chapel от Cray[3].

С ростом числа ядер в массовых системах потребность в ТМ будет только возрастать, и если удастся устранить основные узкие места, то со временем она вполне может стать одной из основных концепций параллельного программирования.

## **7. Литература**

- [1] M. Abadi, A. Birrell, T. Harris, and M. Isard, “Semantics of transactional memory and automatic mutual exclusion”, POPL 2007, pp. 63-74
- [2] Boehm, Hans-J; Adve, Sarita V., “Foundations of the C++ Concurrency Memory Model”, PLDI 2008, pp. 68-78
- [3] B. Chamberlain, “An Introduction to Chapel: Cray's High-Productivity Language”, AHPCRC/DARPA PGAS Conference 2005
- [4] P. Charles, C. Donawa, K. Ebcioglu, C. Grothoff, A. Kielstra, C. Von Praun, V. Saraswat, and V. Sarkar, “X10: An object-oriented approach to non-uniform cluster computing”. OOPSLA, 2005, pp:519-538
- [5] L. Crowl, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum, “Integrating transactional memory into C++”. In TRANSACT 2007, Portland, OR
- [6] L. Dalessandro, V. J. Marathe, M. F. Spear, M. Scott. “Capabilities and limitations of library-based software transactional memory in C++”. TRANSACT 2007.
- [7] D. Dice, M. Herlihy, D. Lea, Y. Lev, V. Luchangco, W. Mesard, M. Moir, K. Moore, D. Nussbaum, “Applications of the Adaptive Transactional Memory Test Platform”, TRANSACT 2008
- [8] AMD, S. Diestelhorst, M. Hohmuth, “Hardware acceleration for lock-free data structures and software-transactional memory”, [http://www.amd64.org/fileadmin/user\\_upload/pub/ephm08-asf-eval.pdf](http://www.amd64.org/fileadmin/user_upload/pub/ephm08-asf-eval.pdf)



- [9] T. Harris and K. Fraser. “Language support for lightweight transactions”. OOPSLA 2003, pp 388-402
- [10] T. Harris, S. Marlow, S. P. Jones and M. Herlihy, “Composable memory transactions”, PPOPP 2005,
- [11] T. Harris, M. Plesko, A. Shinnar, and D. Tarditi. “Optimizing memory transactions”, PLDI 2006. pp 14-25
- [12] IBM, “IBM C/C++ for Transactional Memory for AIX, V0.9, Language Extensions and User’s Guide”, <http://dl.aplhaworks.ibm.com/technologies/xlcstm/xlcstm-whitepaper.pdf>
- [13] Intel, “Intel® C++ STM Compiler Prototype Edition Language Extensions and Users’ Guide”, [http://softwarecommunity.intel.com/isn/Downloads/whatif/stm/Intel-C-STM-Language-Extensions-Users-Guide-V2\\_0.pdf](http://softwarecommunity.intel.com/isn/Downloads/whatif/stm/Intel-C-STM-Language-Extensions-Users-Guide-V2_0.pdf)
- [14] Yang Ni, A. Welc, A.-R. Adl-Tabatabai, M. Bach, S. Berkowits, J. Cownie, R. Geva, S. Kozhukow, R. Narayanaswamy, J. Olivier, S. Preis, B. Saha, , A. Tal, X. Tian, “Design and Implementation of Transactional Constructs for C/C++”, OOPSLA, 2008 (accepted)
- [15] M. F. Ringenburt and D. Grossman. “AtomCaml: first-class atomicity via rollback”. ICFP 2005, pp. 92-104
- [16] Guy Steele, “The Fortress Parallel Programming Language”, GSPx Multi-core Applications Conf. 2006.
- [17] C. Wang, W.-Y. Chen, Y. Wu, B. Saha, and A.-R. Adl-Tabatabai. “Code generation and optimization for transactional memory constructs in an unmanaged language”, CGO 2007, pp. 34-48
- [18] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. ISCA 1995: pp 24-36.

# **Automated Analysis of Cross-Distribution Compatibility of Linux Applications**

<b>Konstantin Vlasov Institute for System Programming Russian Academy of Sciences email: vlasov@ispras.ru</b>	<b>Vladimir Rubanov Institute for System Programming Russian Academy of Sciences email: vrub@ispras.ru</b>	<b>Andrey Smachev Institute for System Programming Russian Academy of Sciences email: biga@ispras.ru</b>
---	--	--

## **Abstract**

This paper presents the Linux Application Checker tool designed for automated analysis of cross-distribution compatibility of Linux applications. The need for such a tool comes from the well-known problem of incompatibilities between Linux distributions that prevent running an application on distributions which it was not designed for. This situation brings problems both to users (because they cannot run applications they need) and developers (because they need to develop “clones” of their applications for different distributions). Linux Application Checker is designed to largely simplify development of portable cross-distribution applications.

The tool analyses the application component external dependencies, takes into account inter-component relations and creates a detailed report with the list of external libraries and interfaces required by the application along with the information about which distributions provide these libraries and interfaces and which do not. In addition, Linux Application Checker executes the tests for compliance to the Linux Standard Base (LSB) standard and provides a simple way for certifying applications.

At the moment, Linux Application Checker is an official tool approved by the Linux Foundation for checking applications for LSB compliance, and it is also recommended to use by any developers interested in writing portable Linux applications.

**Keywords:** Linux, LSB, application portability.

## **Автоматизация анализа совместимости Linux приложений с различными дистрибутивами**

<b>Константин Власов Институт системного программирования РАН email: vlasov@ispras.ru</b>	<b>Владимир Рубанов Институт системного программирования РАН email: vrub@ispras.ru</b>	<b>Андрей Смачёв Институт системного программирования РАН email: biga@ispras.ru</b>
---	--	---

### **Тезисы**

В статье представлен инструмент Linux Application Checker, предназначенный для автоматизированного анализа совместимости приложений с различными дистрибутивами Linux. Разработка приложений, которые могут работать без каких либо модификаций на различных дистрибутивах Linux, чрезвычайно актуальна в условиях наблюдаемого роста популярности данной платформы.

Linux Application Checker анализирует различные компоненты приложения (бинарные модули и скрипты) и выявляет набор внешних зависимостей приложения - главным образом конкретных версий ожидаемых от дистрибутива библиотек и необходимых интерфейсов (функций и глобальных данных). Сопоставляя полученные данные с базой знаний о конкретных дистрибутивах, Linux Application Checker позволяет разработчикам и пользователям приложений определять на каких дистрибутивах данное приложение может быть запущено. Также Linux Application Checker позволяет проводить тестирование приложений на соответствие стандарту LSB.

В настоящее время Linux Application Checker одобрен Linux Foundation в качестве официального средства сертификации приложений на соответствие LSB и рекомендуется к использованию всеми разработчиками, заинтересованными в переносимости их приложений между дистрибутивами Linux.

**Keywords:** Linux, LSB, переносимость приложений.

## Введение

В настоящее время в мире насчитывается более 500 публичных дистрибутивов Linux (например, см. <http://lwn.net/Distributions/>). Каждый из таких дистрибутивов представляет собой уникальную комбинацию определенных версий/модификаций базовых компонентов, таких как ядро, библиотеки, системные утилиты, приложения и т.д. В данном докладе дистрибутивы Linux рассматриваются как платформы для обеспечения функционирования сторонних приложений (то есть не включенных в комплект поставки самим производителем дистрибутива). С этой точки зрения главными компонентами дистрибутива являются собственно ядро операционной системы и набор разделяемых библиотек, которые предоставляют приложениям прикладные бинарные интерфейсы (ABI) в виде функций и глобальных данных.

Проблема заключается в том, что в разных дистрибутивах поставляются разные версии библиотек, в том числе с уникальными изменениями, внесенными разработчиками дистрибутива. В итоге это может означать разные интерфейсы для приложений, как по составу, так и по поведению. Именно поэтому написать стороннее приложение, которое будет работать на всех дистрибутивах, да еще и без перекомпиляции (что важно для многих производителей ПО), оказывается сложной задачей. Производители дистрибутивов стараются смягчить эту проблему, поставляя несколько версий одной и той же библиотеки в составе дистрибутива, чтобы разные приложения могли использовать необходимые им версии. Усилия по стандартизации Linux в виде стандарта Linux Standard Base (LSB) [1]-[2] также способствуют появлению гарантированного набора базовых системных библиотек в основных дистрибутивах.

Однако, разработчикам приложений достаточно сложно найти централизованные сведения о составе различных дистрибутивов для анализа переносимости своих приложений и для принятия решений о возможных конкретных мерах по улучшению этой переносимости. Именно поэтому актуальной задачей становится сбор такой информации и разработка средств автоматизированного анализа переносимости приложений между основными дистрибутивами, сведения о которых имеются в такой базе данных.

В данной статье представлены результаты работ Центра верификации ОС Linux [3] при Институте системного программирования РАН (ИСП РАН) [4], посвященные указанной проблематике. Представлена база знаний о дистрибутивах и инструмент автоматизированного анализа совместимости Linux-приложений с дистрибутивами - Linux Application Checker.

### **База знаний Linux Foundation**

Для стандартизации, защиты и продвижения Linux крупнейшими ИТ-компаниями, среди которых стоит упомянуть IBM, Intel, HP, Novell, Oracle, был создан международный консорциум Linux Foundation [5], который в настоящее время представляет собой основную в мире площадку, объединяющую усилия и экспертизу различных организаций и лиц, заинтересованных в успешном развитии Linux.

Для централизованного анализа экосистемы Linux в рамках программы LSB Infrastructure [6] (совместно Linux Foundation и ИСП РАН) была создана база данных, содержащая информацию о составе различных дистрибутивов (в первую очередь наличие определенных библиотек и интерфейсов), о внешних зависимостях реальных приложений и о стандартизованных в рамках стандарта LSB элементах.

В настоящее время эта база данных включает более 80 таблиц с суммарно 25 миллионами записей. Основную часть данных занимают сведения о дистрибутивах и приложениях. По состоянию на сентябрь 2008 года база содержит информацию о 53 дистрибутивах и 1087 приложениях. Информация постоянно обновляется. Эти сведения используются в том числе для поддержки принятия решений по развитию стандарта LSB.

Содержимое базы данных доступно для удобного просмотра с поиском и кросс-навигацией с помощью специализированной информационной системы LSB Database Navigator [7].


## Linux Application Checker

Чтобы эффективно использовать информацию из построенной базы знаний Linux Foundation для анализа совместимости конкретных приложений с имеющимися в базе дистрибутивами, был разработан инструмент Linux Application Checker (краткое название App Checker), созданный в рамках программы LSB Infrastructure в российском Центре верификации ОС Linux.

Linux Application Checker предоставляет пользователю визуальный веб-интерфейс, который обеспечивается простым веб-сервером, встроенным в инструмент. На стартовой странице (Application Check) пользователю предлагается указать полный набор компонентов тестируемого приложения. В этот набор могут входить следующие данные:

- отдельные файлы;
- каталоги со всем содержимым;
- установочные пакеты и архивы форматов RPM, DEB, TAR.GZ, TAR.BZ2;
- присутствующие в системе установленные пакеты.


В процессе тестирования App Checker просматривает содержимое архивов и каталогов, извлекает файлы, подлежащие проверке (исполняемые двоичные файлы, разделяемые библиотеки, а также скрипты, написанные на языках Shell, Perl и Python) и проводит над ними серию тестов.



THE

LINUX

FOUNDATION



LINUX

APPLICATION

CHECKER

Application Check | **Result History** | Help | About

Administration

Analysis Results for **alsa-1.0.11-32.7** on **x86**

Please **upload** info about your application to the **LSB Navigator**.

Some compatibility problems detected

There are **5** of **28** distributions (see below) that provide all the required libraries and interfaces.

The Application uses **2 external libraries incompatible with LSB 3.2.**

**Distribution Compatibility**

**App Components**

**External Libraries**

**External Interfaces**

**LSB Certification**

The table below shows the compatibility status of your application with the distributions analyzed by the Linux Foundation. Your Application will run on the "green" distributions without loader problems. Compatibility with the "yellow" distributions can be easily achieved by excluding unneeded libraries from the dependencies of your application. Making the Application compatible with the "red" distributions may require more effort to avoid using missing libraries/interfaces or by supplying them as a part of your application package. Please note that functional correctness is not guaranteed by this analysis.

Summary		Missing Libraries		Missing Interfaces		Comments
Distribution	Status	Common	Unknown	Common	Unknown	
SLES 10	OK	none	none	none	none	

Выполняемые тесты можно разделить на две группы: проверка совместимости приложения с различными дистрибутивами Linux

(информация о которых имеется в базе знаний Linux Foundation) и проверка приложения на соответствие стандарту LSB. Тесты из первой категории предназначены для всех разработчиков, перед которыми стоит задача обеспечения работоспособности их приложения на различных Linux платформах. Пользователю выдаётся набор отчётов, в которых перечислены проблемы, не позволяющие использовать приложение на определенных дистрибутивах. Самой часто встречающейся проблемой является отсутствие необходимых версий библиотек или интерфейсов в тех или иных дистрибутивах. App Checker считывает из ELF-файлов секции `.dynamic` и выделяет оттуда записи типа `DT_NEEDED`, в которых и перечислены все библиотеки, необходимые для запуска приложения. Также анализируются секции `.dynsym` и `.symtab`, откуда извлекается информация о необходимых приложению внешних ELF-символах (интерфейсах). Разумеется, при анализе учитываются все внутренние зависимости между компонентами приложения, и пользователю выдаётся только список внешних библиотек и интерфейсов, которые не содержатся в компонентах самого приложения.

В дополнение к списку зависимостей App Checker выводит, где возможно, различные сведения и рекомендации из базы знаний Linux Foundation, например рекомендации, чем заменить устаревшие (deprecated) интерфейсы, или советы включить определенную библиотеку в состав поставки приложения.

LSB-тесты более специфичны. Их основное назначение — анализ расхождений со стандартом LSB с целью их устранения, а также создание отчёта, на основе которого сертификационная комиссия сможет вынести заключение о соответствии (или несоответствии) приложения стандарту LSB. Анализ требуемых библиотек и интерфейсов здесь также выполняется, но полученный список сравнивается не с базой дистрибутивов, а с информацией о стандартизованных LSB элементах (приложение не должно использовать библиотеки и интерфейсы за пределами стандартизованного LSB подмножества). Помимо этого проверяются другие аспекты, регламентируемые LSB, такие как бинарная структура ELF-файлов и пакетов RPM, использование специального LSB-загрузчика, и т. д.

### **Использование отчетов Linux Application Checker**

По окончании тестирования Linux Application Checker выводит отчёт, общий вид которого представлен на рис. 1. В верхней части находится краткий вердикт результатов тестирования, которое может либо пройти успешно, либо сообщить о частичной совместимости, либо выдать вердикт о полной несовместимости приложения с известными

дистрибутивами. Чуть ниже можно видеть сводку (summary) по основным аспектам совместимости, таким как количество дистрибутивов, на которых приложение запустится без проблем, или количество используемых нестандартных библиотек и интерфейсов. Далее расположены пять вкладок-отчетов, на каждой из которых можно найти детальную информацию, имеющую отношение к проблемам совместимости. Рассмотрим содержимое этих вкладок более подробно.

## **Отчет Distribution Compatibility**

Эта вкладка открыта по умолчанию и содержит краткую сводку о том, с какими дистрибутивами из числа присутствующих в базе Linux Foundation приложение совместимо, а с какими — нет. Информация представлена в виде таблицы, где каждая строка соответствует конкретному дистрибутиву и окрашена в зеленый, оранжевый или красный цвет согласно статусу совместимости. Важно понимать, что анализ на совместимость гарантирует лишь возможность запуска приложения, что не всегда означает корректную работу. Основную часть анализа составляет проверка наличия в дистрибутиве всех библиотек и интерфейсов, необходимых для работы приложения. Наведя указатель мыши на ячейку со статусом для конкретного дистрибутива, можно увидеть количество недостающих библиотек и интерфейсов, а щёлкнув по любой такой ячейке — развернуть таблицу, отобразив эти данные сразу для всех дистрибутивов. Стоит отметить, что существует особая категория библиотек «Unknown»: в неё входят те элементы, о наличии или отсутствии которых в дистрибутиве ничего не известно. Такая ситуация вызвана тем, что хранение всего содержимого для каждого дистрибутива слишком сильно увеличило бы базу данных и затруднило её использование. Поэтому была выделена специальная группа так называемых «известных» разделяемых библиотек, в которую включаются библиотеки, которые могут включаться в состав дистрибутивов для совместного использования различными приложениями. Это позволяет исключить хранение библиотек, присутствующих в дистрибутиве исключительно как часть какого-то самостоятельного приложения, и тем самым значительно уменьшить размер запоминаемых данных. В настоящее время список включает 723 записи soname «известных» библиотек и продолжает постоянно пополняться по мере выхода новых версий.

Таким образом, первая страница отчёта позволяет быстро определить, какие дистрибутивы предоставляют все необходимые приложению библиотеки и интерфейсы, а для проблемных дистрибутивов узнать (посредством гиперссылок, открывающих дополнительные



информационные окна), чего именно не хватает в конкретном дистрибутиве.

Помимо вышеперечисленного, существует ещё один аспект анализа, который не является тестом на совместимость, но который может заметно облегчить процедуру портирования приложения. В силу различных обстоятельств нередко происходит так, что в список зависимостей приложения включена библиотека, из которой не используется ни одного интерфейса. App Checker отслеживает такие ситуации и сообщает о наличии «ненужных» библиотек. Разработчик,

Рис. 2. Вкладка App Components

обладая этой информацией, сможет подправить опции компоновщика так, чтобы эти неиспользуемые библиотеки не подключались, и, таким образом, практически без трудозатрат улучшить совместимость с теми дистрибутивами, на которых этих библиотек нет.

Отчет App Components

Следующая вкладка предоставляет детальные сведения о протестированных компонентах приложения (см. рис. 2). К таким компонентам относятся отдельные бинарные модули, разделяемые библиотеки и различные скрипты, входящие в поставку приложения. Для каждого такого компонента выводятся сведения о степени его совместимости с дистрибутивами и со стандартом LSB (зависимости от других компонентов приложения исключаются) с указанием количества конкретных проблем (отсутствующих библиотек и интерфейсов). Это позволяет быстро идентифицировать проблемные компоненты в составе больших приложений. По ссылкам в ячейках таблицы можно перейти к более детальной информации.

Distribution Compatibility   App Components   External Libraries   External Interfaces   LSB Certification						
Component	Distribution Compatibility	Missing Libraries	Missing Interfaces	Unneeded Libraries	LSB 3.2 Compatibility	Comments
PKG:/usr/bin/acconnect	26 of 28 (list .)	none	3 of 14 ( . )		3 interfaces are since LSB 4.0	
PKG:/usr/bin/alsamixer	22 of 28 (list .)	1 of 5 ( . )	25 of 45 ( . )		1 non-LSB library, 6 non-LSB interfaces	

Отчет External Libraries

На третьей вкладке, изображённой на рис. 3, представлены данные об используемых приложением библиотеках. Для каждой библиотеки приводятся следующие сведения:

- список компонентов приложения, которым требуется указанная библиотека;
- количество интерфейсов, используемых приложением из данной библиотеки;
- дистрибутивы, в которых библиотека присутствует;

- статус библиотеки по отношению к стандарту LSB.

Distribution Compatibility
App Components
External Libraries
External Interfaces
LSB Certification

The table below lists all external libraries required by your Application (based on DT\_NEEDED ELF section). LSB and Distribution presence statuses are provided for each library. You can adjust distributions of interest and the report will be updated appropriately.

Select distributions... (28 of 28 selected)

Component: All
Show: All
6 libraries shown. [Reset all filters]

Library	Dependent Application Components	Interfaces Actually Used	Presence in Distributions	In LSB 3.2?	More Info
libc.so.6	26 components (...)	114	28 of 28 (list...)	Yes	Details...
libdl.so.2	20 components (...)	4	28 of 28 (list...)	Yes	Details...

При помощи фильтров пользователь может настроить отображение только интересующих его сведений. В частности, можно ограничить список анализируемых дистрибутивов, например, исключив все устаревшие (выпущенные ранее определённого года), или показав только библиотеки, имеющие проблемы совместимости.

## Отчет External Interfaces

Данная вкладка практически полностью аналогична предыдущей, только предоставляет сведения на уровне интерфейсов вместо библиотек. В число отображаемых данных добавляются версии интерфейсов и имена библиотек, экспортирующих указанные интерфейсы.

## Отчет LSB Certification

Эта вкладка создана для облегчения процесса сертификации приложения на соответствие стандарту LSB. Пользователю выводится список всех обнаруженных проблем, который можно группировать либо по компонентам, либо по категориям ошибок. Если приложение соответствует LSB, разработчик может начать процедуру сертификации, перейдя по соответствующей ссылке в онлайн сертификационную систему Linux Foundation. Ему будет предложено создать учётную запись (если он не делал этого ранее), зарегистрировать продукт и загрузить результаты тестирования, подготовленные App Checker. В случае успешной сертификации приложение будет включено в официальный реестр LSB-сертифицированных продуктов.

## Загрузка данных на сервер Linux Foundation

Даже если приложение не проходит тест на LSB, информацию о нём можно загрузить в базу данных Linux Foundation. Это помогает анализировать реальные потребности разработчиков программ и учитывать их при обсуждении состава будущих версий стандарта LSB. На странице результатов присутствует специальная ссылка «upload», при щелчке по которой открывается форма загрузки. В ней требуется указать имя приложения, его версию, некоторые данные о типе приложения и его характеристиках, после чего информация о составе приложения и его зависимостях будет загружена на FTP-сервер Linux Foundation, откуда она будет далее внесена в базу данных. Данные о загруженных таким образом приложениях можно видеть в LSB Database Navigator по ссылке <http://linuxfoundation.org/navigator/browse/app.php>.

## Заключение

Различия в дистрибутивах Linux на уровне предоставляемых приложениям операционной системой сервисов, библиотек и функций определенно создают проблемы для независимых разработчиков приложений, которым часто приходится создавать отдельные сборки их программ для каждого из основных дистрибутивов. К счастью, инициативы по стандартизации платформы Linux и усилия производителей дистрибутивов по поддержке унаследованных приложений приводят к появлению определенного базиса, на наличие которого можно рассчитывать в большинстве популярных дистрибутивов. Приложения, использующие возможности только этого базиса, становятся вполне переносимыми между различными дистрибутивами даже на бинарном уровне.

Ведущие мировые ИТ-компании в рамках консорциума Linux Foundation прилагают серьезные усилия для облегчения поддержки Linux производителями независимых приложений. Представленные в данной статье база знаний и инструмент Linux Application Checker, разработанные по заказу Linux Foundation российскими специалистами, являются в настоящее время основными инфраструктурными компонентами, позволяющими разработчикам Linux приложений эффективно анализировать и обеспечивать переносимость своих приложений. Также, Linux Application Checker является официальным средством автоматизированной сертификации приложений на LSB-совместимость. Последнюю версию инструмента можно загрузить со страницы проекта LSB Infrastructure [6].

## Ссылки

- [1] Linux Standard Base Homepage.  
<http://www.linuxfoundation.org/en/LSB/>.
- [2] V. Rubanov. Linux Standard Base (LSB): “Single Linux”  
Specification and Support Infrastructure.  
In Proceedings of SECR 2007.
- [3] Центр верификации ОС Linux. <http://linuxtesting.org/>
- [4] Институт системного программирования РАН. <http://ispras.ru/>
- [5] Linux Foundation.  
<http://linuxfoundation.org/>
- [6] LSB Infrastructure Program.  
<http://ispras.linuxfoundation.org/>
- [7] LSB Database Navigator.  
<http://linuxfoundation.org/navigator/>

# Задача как мотивирующий фактор

Светлана А. Савельева  
AT Software  
s.saveljeva@at-software

... один программист кладет перед другим однострочную программу и либо гордо рассказывает, что она делает, а затем спрашивает: "А ты можешь закодировать это меньшим количеством символов?" - как будто это имеет какую-то практическую ценность, - либо просто спрашивает: "Угадай, как это работает!".

**Эдсгер В. Дейкстра, «Смирненный программист»**

## **Вступление.**

Удержание квалифицированного разработчика ПО в компании при современной ситуации на рынке труда является одной из наиболее актуальных задач в сфере управления персоналом. Если еще несколько лет назад средний срок работы на одном месте для квалифицированного разработчика оставлял порядка 3-5 лет, то сейчас, по оценкам представителей hr-служб, смена места работы раз в 1-2 года считается нормой.

Проблемой удовлетворенности трудом мотивации занимались такие ученые, как Ф. Тейлор, А. Файоль, Э. Мейо, Г. Форд, А. Маслоу, Д. Мак-Грегор, К. Альдерфер, Д. Мак-Клелланд, Ф. Херцберг, Л. Портер, Б.Ф. Скиннер и др. Эти ученые создавали различные теории и модели, выработали практические рекомендации, широко применяющиеся в современной практике управления персоналом.

Изучая вопрос мотивации и удовлетворенности трудом в IT-сфере, написали свои работы такие авторы как Джоэл Спольски, Эдсгер Дейкстра, Эдвард Йордон и ряд других авторов.

Безусловно, профессиональная деятельность разработчика ПО имеет свою специфику с точки зрения факторов, определяющих уровень удовлетворенности трудом и, как следствие, уровень лояльности к работодателю. Если сравнивать так называемые «карты мотиваторов» среднестатистического разработчика ПО и, например, таковые менеджера по продажам, мы получим совершенно разные картины: активность и заинтересованность разработчика и продавца определяют различные факторы.

На практике, анализируя результаты job interview, и в частности, причины, по которым разработчик покидал предыдущие проекты или компании по собственному желанию, можно выделить три основные. Чаще всего ими являются:

- неудовлетворительное качество менеджмента проекта
- неудовлетворительная оплата труда
- неинтересная задача

Порядок с точки зрения приоритетности в каждом отдельном случае может быть различным, но все три причины, в большинстве случаев, так или иначе, озвучиваются в процессе прохождения интервью при найме.

Ниже мы более подробно рассмотрим один из перечисленных факторов, определяющий удовлетворенность трудом разработчика ПО и качество выполняемой работы, а именно характер выполняемой задачи и, как следствие, **отношение разработчика к задаче**.

Для получения качественного ПО важно, чтобы разработчик принимал задачу, чтобы она была ему интересна и соответствовала его личностным особенностям.

Небольшое замечание: у нас нет точных статистических данных, но эмпирический опыт показывает, что чем выше квалификация разработчика, тем более значим для него фактор «интересности» задачи при принятии того или иного предложения о найме.

## 1. Какими могут быть задачи?

1. Если попытаться описать характер существующих задач, то можно получить ту или иную типологизацию. Например, при классификации задач можно выделить следующие критерии:
  - а) Степень самостоятельности при выполнении задачи (от реализации небольшой готовой задачи до

- самостоятельного создания архитектуры и декомпозиции для других участников команды)
- b) Предметная область в зависимости от функциональной части системы (GUI, базы данных, и пр.)
  - c) Коммуникация с клиентом (общается напрямую с заказчиком или нет)
  - d) Степень «рутинности» задачи (монотонная однообразная задача или поддержка; Start-up's; исследования или инновации; регулярная смена задач или даже технологий)

Безусловно, это далеко не единственные возможные критерии, на основании которых можно сделать те или иные описания. Но их, на наш взгляд, можно зафиксировать как одни из наиболее «ежедневных», встречающихся в повседневной практике ведения проекта по разработке ПО.

На основе подобной классификации могут быть составлены небольшие опросники, которые могут быть использованы для более четкого понимания предпочтения разработчиков в отношении задачи (в качестве примера может быть представлен опросник, данный в приложении 2). Такие методы исследования позволяют дать прогноз на поведение разработчика в проекте в отношении принятия задачи и эффективности ее реализации. Они показывают, какая задача будет для разработчика наиболее «приемлемой» и где его труд будет наиболее эффективен.

Это позволяет более качественно и целенаправленно осуществлять управление человеческими ресурсами при разработке ПО.

## **2. Отношение разработчика к задаче и практика hr-менеджмента**

Если говорить об общих процессах управления человеческими ресурсами компании (human resources), то, на наш взгляд, важно учитывать фактор принятия задачи и уделять отдельное внимание подбору задачи в соответствии с ожиданиями и предпочтениями разработчика. Нужно стараться привести в соответствие «вкусы» разработчика и тип задачи, которую он получает для реализации (хотя далеко не всегда это возможно). Если этого не происходит, у разработчика довольно часто снижается производительность труда, он делает меньше работы, а ее качество становится хуже. На фоне утраты интереса к задаче происходит снижение лояльности к

работодателю, и, как следствие, повышается «текучка», что автоматически снижает стоимость компании

На наш взгляд, важно выяснить ожидания и предпочтения разработчика в следующие моменты его работы в компании

- на стадии первичного интервью при найме.

Приглашая нового сотрудника в компанию, нужно четко понимать, под какую задачу он берется и в какой степени эта задача соответствует его ожиданиям.

- На стадии включенности в проект.

Регулярный мониторинг на предмет удовлетворенности трудом во время проекта позволяет прогнозировать уход и «играть на опережение»

- На моменте перерыва в работе при переходе в новый проект

Здесь нужно быть особенно внимательным, так как осознание собственного отношения разработчика к задаче в данном случае особенно велико и потребность в соответствии своим ожиданиям наиболее актуализирована. Т.е. это тот момент, когда у разработчика, в силу возникшей паузы, есть возможность подумать не только о задаче, но и о себе.

Реализация подобных практик может быть осуществлена через внутренние документы, фиксирующие процессы службы персонала. Могут быть разработаны и утверждены графики мониторинга, исследования можно привязывать к регулярной аттестации, если таковая существует в компании, необходимость проведения периодического опроса на предмет удовлетворенности задачей может быть зафиксирована в положении о персонале и т.д.

### **3. Задача и управление ожиданиями клиента.**

При аутсорсинговой (сервисной, а не продуктовой) модели бизнеса, проблема качества и характера задачи, над которой будет работать девелопер, встает особенно остро. Задачи, а иногда и состав команды, утверждает непосредственно заказчик, лишая руководителя проекта возможности свободно выбора задачи для того или иного программиста или произвольной ротации.

Соответственно, полноценный процесс управления человеческим капиталом в такого рода бизнесе включает в себя и аспект управления ожиданиями клиента. Клиент должен осознавать, что однообразная и неинтересная задача снижает мотивацию и



удовлетворенность трудом у команды. Довольно часто, по нашей практике, со стороны заказчика высказываются опасения относительно любых перестановок внутри команды, а увольнение разработчика по собственной инициативе рассматривается как настоящая проблема. Но, на наш взгляд, одной из основных задач, лежащих на стыке hr- и клиент-менеджмента, являются выстраивание осознанных ротаций внутри проектной группы и вовлечение в обсуждение этого вопроса клиента. Этот процесс должен быть плановым и управляемым. Перемещения, вводы и выходы людей из команды определяются заблаговременно, обсуждаются и согласовываются с заказчиком. До клиента важно донести, что стратегия простого «силового» удержания разработчика в течение долгого времени на одной и той же задаче, которая данному разработчику не интересна, ведет к снижению мотивации, ухудшению качества создаваемого кода и, в конечном итоге, потере разработчика как участника команды.

Понятие «развития команды» в данном случае включает в себя с одной стороны, плановое изменение в характере задач путем обсуждения таких возможностей с заказчиком, с другой стороны, с плановыми перестановками внутри команды в соответствии с ожидаемыми задачами.

Вполне понятно, что «интересных задач» на всех не хватает, и, осуществляя аутсорсинг, заказчик зачастую пытается вынести за пределы своей компании как раз наиболее рутинные и скучные, с точки зрения выполнения, работы. Но, как показывает наш опыт, если клиент настроен на долгосрочное сотрудничество, он готов к обсуждению такого рода вопросов и к конструктивным действиям в этом направлении.

## **Заключение.**

Таким образом, повышение удовлетворенности трудом у developer's с целью сохранения его мотивации удержания в компании требует (помимо прочего):

- Диагностики и понимания предпочтения разработчика в отношении задачи.

Важно адекватно и тщательно провести ее еще на моменте найма нового сотрудника в компанию и учитывать его особенности при дальнейшем распределении в проекты. Диагностика может быть осуществлена различными методами, от неформальных бесед

общего характера до использования специально разработанных тестовых методик и опросников.

- Регулярного мониторинга удовлетворенности разработчика задач.

Безусловно, сам по себе такого рода мониторинг несет определенный риск: его проведение с большой вероятностью ведет к осознанию разработчиком неудовлетворенности задачей и актуализацию потребности в смене задачи, а работодатель не всегда имеет возможность предложения интересной задачи, интересных задач на всех не хватает. Поэтому, планируя мониторинг, нужно помнить о возможности возникновения подобной проблемы. Но, не смотря на то, что такого рода диагностика несет определенную опасность, ее полное отсутствие так же может обернуться возникновением сложностей. Т.е. решение о проведении мониторинга либо об отказе от него должно быть комплексным и взвешенным, учитывающим все влияющие факторы и возможные последствия.

- Регулярной ротации в случае неудовлетворенности задачей
- Простроенных (проактивных) отношений с клиентом в смысле обсуждения качества предлагаемых задач, а так же выведения «уставших» разработчиков в другие проекты.

### **Приложение 1.**

Ниже мы приведем несколько примеров из практики, которые показывают, насколько важны понимание и учет ожиданий разработчика в отношении предложенной ему задачи и как это влияет на ход проекта в целом.

### **Пример 1.**

В одном из проектов при работе с западным заказчиком, надо сказать, весьма чувствительным к выводам людей из проекта, один из участков работы, рассчитанный на одного разработчика, постоянно являлся «камнем преткновения» с точки зрения «текучки». После небольшого срока работы, обычно после 2-3 месяцев, разработчики отказывались от выполнения данной работы с просьбой перевести их на другую задачу или проект. При внимательном рассмотрении вопроса выяснилось, что предложенная заказчиком архитектура, в рамках которой приходилось работать нашим девелоперам, была весьма специфичной и «противоестественной» с точки зрения большинства современных программистов. Она была разработана заказчиком довольно давно и в силу внутренних причин сохранялась в этой части проекта, хотя в остальных частях системы уже была изменена. Учитывая важность заказчика для компании, был проведен серьезный поиск подходящего человека, который был бы готов работать в таких условиях. Т.е., требовался специалист, имеющий достаточную толерантность к «неудобной» архитектуре, готовый выполнять мелкие четко ограниченные заказчиком задачи, не имея при этом права на самостоятельные технические решения за пределами поставленных задач. По результатам полученного «портрета» был найден соответствующий разработчик. Не смотря на то, что его поиск занял довольно много ресурсов, в конечном итоге, проблема «текучки» на данном участке работ была решена, а отношения с заказчиком сохранены на должном уровне.

### **Пример 2.**

#### **Start-up's**

При прохождении exit-interview один из разработчиков, обсуждая причины своего увольнения, в качестве основной причины назвал частое привлечение его к start-up's новых проектов. Менеджер, проводивший интервью, был крайне удивлен, т.к. увольняющийся сотрудник всегда показывал хороший результат и считался вполне успешным разработчиком в «амплуа» такого рода. В процессе разговора стало понятно, что, не смотря на то, что девелопер был успешен в подобной деятельности и у него это хорошо получалось, сам он крайне тяжело переживал такие ситуации. Новые проекты давались ему ценой большого напряжения и переживались как серьезный стресс. Поскольку, как уже было сказано, результат он

демонстрировал вполне успешный, ни менеджер проекта, ни служба персонала вовремя не поинтересовались глубинной сутью происходящего. Не исследовались общая удовлетворенность трудом, не обсуждались те или иные предпочтения в отношении задачи а так же в отношении роли в проекте. Присутствие некоторых особенностей личности, и, как следствие, негативное отношение к участию в start-up's, выяснилась только на exit-interview. Разработчик, как сотрудник компании, был потерян.

### **Пример 3.**

Небольшой проект по разработке ПО для западной телекоммуникационной компании отличался на редкость рутинной задачей и крайне высокими требованиями к уровню профессионализма девелоперов со стороны клиента. Требования к квалификации разработчиков были явно завышенными, но, тем не менее, такие разработчики были заказчику предоставлены. С одной стороны, желание клиента иметь как можно более квалифицированный персонал для выполнения своих работ вполне объясним и понятен. С другой стороны, столь явное несоответствие этого высокого уровня довольно простому характеру задачи привело к ряду проблем в проекте. Через полгода от момента начала работ, из проекта, по собственному желанию, вышел первый человек, а через год ситуация начала ухудшаться, т.к. о своем желании уйти заявили еще несколько разработчиков. Все это время, начиная от момента start-up, с клиентом велись разговоры о том, что его требования к квалификации явно завышены и ситуация over qualified рано или поздно даст о себе знать. Но переговоры не привели к какому-либо решению вопроса. В результате удалось достигнуть только договоренности о смене всей команды, к счастью, она не была большой, но при этом новая команда должна была иметь такую же высокую квалификацию, как и предыдущая, не смотря на все ту же простую задачу. Очевидно, что взаимодействие с клиентом в данном случае нуждается в улучшении и дальнейших переговорах.

### **Пример 4.**

В компании в течение длительного времени существовал большой проект. Довольно многочисленная группа разработчиков высокой квалификации работала практически в одном составе на одного заказчика в течение нескольких лет. Задачи всегда были однородными и похожими одна на другую, менеджеры со стороны заказчика не менялись. Когда по объективным причинам проект был закрыт, компания, рассчитывая на освобождение серьезного

трудового ресурса, столкнулась со следующей проблемой: за время долгой работы над одним проектом, разработчики, не смотря на свою высокую квалификацию, потеряли навык адаптации и вхождения в новый проект. Первые попытки запуска проекта этой группой оказались неудачными. Отсутствовал навык start-up'a. Пришлось разделять группу на отдельные части или выводить отдельных людей и включать их в существующие проекты. Таким образом, выполнение долгой однородной задачи и отсутствие действий по ротации разработчиков привело к общей своеобразной деформации квалификации (фактически, к снижению) и невозможности дальнейшего использования всей группы как единой команды.

## **Приложение 2.**

Ниже находится опросник, который можно использовать для исследования предпочтений в области задачи в повседневной менеджерской практике. На наш взгляд, рабочий инструмент для исследования предпочтений разработчика должен быть крайне прост в заполнении и однозначен в интерпретации. Сложные развернутые опросники не будут, на наш взгляд, иметь успеха, т.к. ни технические специалисты, ни менеджеры не мотивированы сами по себе на выполнение больших сложных заданий, цели и смысл которых для них не очевидны.

Среднее время ответа на список вопросов, представленных ниже, составляет порядка 10 мин.

Как правило, мы выдаем подобный опросник на первичном интервью при найме на открытую позицию.

Опросник для выяснения профессиональных предпочтений разработчика ПО.

1. Свое дальнейшее профессиональное развитие Вы видите как менеджер проектов или технический гуру?
2. Какие технологии разработки ПО Вам будут наиболее интересны в ближайшие 2-3 года?
3. Какая предметная область в зависимости от функциональной части системы Вам наиболее интересна?
  - a) GUI
  - b) DB
  - c) Системный уровень
  - d) Что-то еще

4. Как Вы оцениваете наиболее комфортную для Вас степень предоставленной самостоятельности при принятии решения?

- a) получить небольшие готовые задачи
- b) реализовывать определенный функционал, самостоятельно прописывая конкретные задачи
- c) заниматься декомпозицией
- d) архитектурой системы в целом

5. Коммуникация с клиентом.

- a) готов общаться с клиентом напрямую по любым вопросам
- b) готов общаться только в рамках своей узкой задачи
- c) хотел бы вести все общение только через менеджера проекта или team-lead
- d) не важно
- e) что-то еще

6. «Рутинность» деятельности.

Предпочитаю:

- a) Однообразная стабильная задача
- b) Поддержка (maintenance)
- c) Разнообразные задачи
- d) Start-up's
- e) исследования и разработки
- f) не важно
- g) что-то еще

7. Отношение к «чужому» коду.

- a) готов работать
- b) предпочел бы не работать
- c) что-то еще

8. Какой процент повседневных задач, на Ваш взгляд, должен содержать «вызов интеллект», т.е. требовать от Вас

приложения серьезных интеллектуальных усилий для их решения? \_\_\_\_\_

Спасибо за заполнение анкеты, мы надеемся, что Ваши ответы позволят сделать наше сотрудничество более эффективным.

#### Список литературы:

1. Дейкстра Э., «Смирный программист», 1972
2. Йордон Э., «Путь камикадзе», Москва, Лори, 2008
3. Рейнвотер Х., «Как пасти котов. Наставление программистам, руководящим другими программистами», СПб, 2006
4. Спольски Дж., «Джоэл о программировании" Символ-плюс, Санкт-Петербург, Москва, 2006
5. Шелдрейк Дж. Теория менеджмента: от тейлоризма до японизации / Пер. с англ. под ред. В.А.Спивака. - СПб: Питер, 2001

# **System Engineering: from marketing requirements to product specifications**

**Larisa Melikhova**  
**System Engineer,**  
**Motorola St. Petersburg**  
**Software Center**  
**Email: alm039@mot.com**

**Dmitry Vavilov**  
**System Engineer, Group**  
**Leader,**  
**Motorola St. Petersburg**  
**Software Center**  
**email: adv008@mot.com**

## **Abstract**

Common definition of system engineer's work is "executing an interdisciplinary process to ensure that customer and end user's needs are satisfied". In Russia such job position is not widespread, besides there is an ambiguity associated with the Russian term, so this was an innovation of Motorola to create in St.Petersburg Software Development Center a team of system engineers who participate in developing software for a Digital TV device. The role of system engineers is receiving market requirements as an input, and providing developers, testers and software architects with clear and unambiguous technical requirements and design specifications, including use cases and feature description for the entire system. The paper shows how these goals can be achieved in the framework of 3-level model for requirements development with reusability principles to be followed. Theoretical considerations are illustrated by examples of how it was applied in a cable TV environment.

**Keywords:** System Engineer, requirements, software development.



## **Работа системного инженера: от маркетинговых требований к спецификациям продукта**

**Лариса Мелихова**  
системный инженер,  
СПб Центр разработки  
программного обеспечения  
**Motorola**  
**Email: alm039@mot.com**

**Дмитрий Вавилов**  
системный инженер, рук.  
группы,  
СПб Центр разработки  
программного обеспечения  
**Motorola**  
**Email: adv008@mot.com**

### **Тезисы**

Обычное определение работы системного инженера – выполнение междисциплинарного процесса, обеспечивающего удовлетворение потребностей как заказчика, так и конечного пользователя. В России эта должность пока мало распространена, к тому же существует неоднозначность в понимании самого термина «системный инженер». Инновационный подход компании Motorola состоит в том, что в Санкт-Петербургском Центре обеспечения Motorola была создана группа системных инженеров, участвующих в разработке программного обеспечения для устройств цифрового телевидения. Роль группы состоит в том, чтобы, получив на входе маркетинговые требования, предоставить разработчикам, тестировщикам и конструкторам программного обеспечения ясные и непротиворечивые технические требования и дизайн-спецификации, включающие набор сценариев использования и описание функциональности для всей системы. В статье показано, каким образом достигаются поставленные цели в структуре 3-уровневой модели разработки требований с соблюдением принципа переиспользования требований. Теоретические рассуждения проиллюстрированы примерами работы в области кабельного телевидения.

**Ключевые слова:** системный инженер, требования, разработка программного обеспечения.

### **1. Введение**

На заре развития информационных систем программисты осуществляли все стадии процесса разработки. Однако, по мере роста и усложнения программного обеспечения, структура процесса разработки претерпела изменения: появились новые участники процесса – тестировщики, системные архитекторы, дизайнеры. Важным звеном современного процесса разработки

является системный инженер: его задача состоит в том, чтобы обеспечить выполнение междисциплинарного процесса, при котором потребности заказчиков и конечных пользователей своевременно удовлетворяются при соблюдении надлежащего качества, надежности, экономичности. Работа системного инженера сопровождает процесс создания продукта на протяжении всего цикла развития системы, от разработки до продажи.

## **2. Предпосылки**

Рабочий процесс в Санкт-Петербургском Центре разработки программного обеспечения Motorola за последние годы претерпел значительные изменения. Если 10 лет назад этот процесс включал только основных участников (менеджмент, разработчиков, тестировщиков, etc.), то со временем возникла группа качества SQA (не путать с группой тестирования) и группа процесса, которая вплотную занялась разработкой самого процесса разработки, что позволило поднять качество разрабатываемой продукции на другой уровень.

Важной составляющей процесса разработки является разработка требований. До недавнего времени существовал процесс создания требований для проекта, который можно условно разбить на 2 фазы. На первой фазе разработчики получали «снаружи» (от коллег, работающих в контакте с заказчиком) маркетинговые требования, и на их основе создавали технические требования. На втором шаге те же разработчики использовали технические требования для создания спецификаций продукта, или PFS – Product Functional Specification. На основе PFS тестировщики разрабатывали тесты, сопоставляя сами тесты с элементами PFS, а наборы тестовых данных – с техническими требованиями.

По мере интенсификации и усложнения задач усложнился процесс разработки. Сегодня не только маркетинговые требования разрабатываются вне Петербургского центра, но и задача создания пользовательского интерфейса, как правило, передается третьей стороне. В то же время, была выделена специальная группа системных инженеров, задача которых – создание технических требований. Таким образом, системные инженеры Центра выполняют роль промежуточной инстанции между заказчиками (или отделом маркетинга) и разработчиками.

Данная группа сфокусирована на том, чтобы в процессе изучения предметной области, анализа маркетинговых требований и общения с разработчиками и тестерами, во-первых, приобрести системное представление о продукте, и во-вторых, детально, непротиворечиво, полно и формально описать это представление.

### **3. Разработка требований**

#### **3.1. Три точки зрения на требования**

Существует по крайней мере три точки зрения на требования, которые следует принимать во внимание при их разработке. Они выражены следующими участниками процесса:

1. Заинтересованная сторона. Это группа, задача которой состоит в том, чтобы определить высокоуровневые требования. Она может включать заказчиков, пользователей, менеджмент, и т.д.
2. Разработчики, задача которых – сделать продукт, описанный в требованиях, на основе системных требований, описывающих логику поведения системы.
3. Тестировщики, которые должны проверить соответствие намерений заказчика и интерпретации разработчиков в виде выпущенного продукта.

#### **3.2. Трехуровневая модель требований**

Модель, взятая за основу в Санкт-Петербургском Центре разработки, включает три уровня требований.

**Маркетинговый уровень.** На это уровне создаются требования, которые называются либо MR – Marketing Requirements, либо FR – Functional Requirements. Требования маркетингового уровня описывают функциональность устройства (например, наличие функции записи телевизионных программ и их последующего воспроизведения) Эти требования разрабатываются группой маркетинга в сотрудничестве с заказчиком.

**Системный уровень.** На этом уровне создаются технические требования, или TR (Technical Requirements). Это устойчивые требования, которые не меняются от одной модели к другой и скорее всего останутся неизменными в ближайшем будущем. Эти требования описывают существенное поведение системы, которое позволяет системе выполнять свои функции (например,

режимы воспроизведения записи телевизионных программ). Условия, при которых требование относится к системному уровню, включают его независимость от конкретной реализации и возможность тестирования в модели "черного ящика", которая не содержит информации о внутренней структуре и дизайне продукта. Можно сказать, что системные требования описывают логику поведения системы, не вдаваясь в детали разработки той или иной модели.

**Требования реализации.** Эти требования входят в уже упоминавшийся набор PFS – Product Functional Specifications. Требования этого уровня могут изменяться от одной модели к другой, поскольку процесс разработки постоянно совершенствуется. Требования реализации (связанные с системными) описывают конкретную реализацию продукта и включают различные детали, например, параметры производительности, потребление электроэнергии, временные характеристики и пр.

### **3.3. Роль системного инженера**

Таким образом, основная задача системного инженера, занятого в проекте разработки программного обеспечения, состоит в разработке требований системного уровня – TR, являющихся связующим звеном между маркетинговыми требованиями и спецификациями на продукт. Решение этой задачи связано с выполнением целого ряда функций:

1. Сбор информации по данной предметной области, включая изучение стандартов и спецификаций.
2. Анализ дизайна разрабатываемой системы для решения вопросов, связанных с функциональностью системы.
3. Анализ архитектуры системы для создания описаний и данных, необходимых разработчикам.
4. Разработка пользовательского интерфейса, предоставляемого дизайнерам.
5. Обсуждение требований с экспертами в данной предметной области, а также с разработчиками и тестировщиками.

Заметим, что эволюция информационных технологий сегодня позволяет использовать виртуальные распределенные группы системных инженеров: именно этот подход применен в Петербургском Центре Motorola.

Выделение специальной группы системных инженеров имеет

следующие преимущества:

1. Узкая специализация инженеров позволяет им более эффективно собирать и обрабатывать информацию.

2. Системные инженеры играют координирующую роль, работая со специалистами из разных групп: маркетинг, дизайнеры (User Interface), разработчики, тестировщики.

3. Именно системные инженеры имеют возможность рассмотреть систему в целом, в то же время фокусируясь на интересах пользователя.

## 4. Примеры

Ниже представлены некоторые примеры из практической области работы группы системных инженеров в Санкт-Петербургском Центре разработки программного обеспечения Motorola.

### 4.1. Соотношение количества требований

На рисунке 1 представлены примеры соотношения различных типов требований: MR/TR/PFS для двух предметных областей. Из приведенных примеров видно, что соотношение примерно одинаковое: маркетинговые требования составляют от 10 до 20% технических требований, в свою очередь технические требования составляют 5—60% от количества требований реализации.

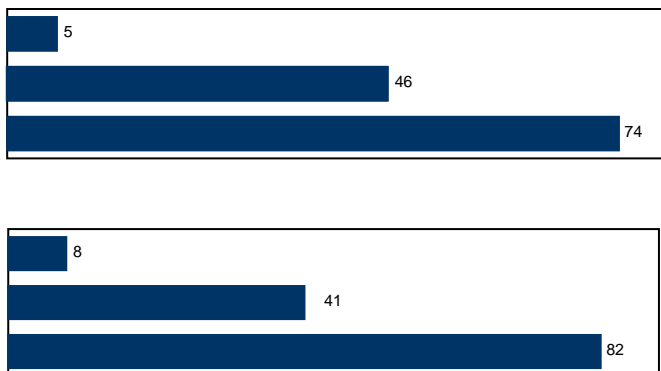


Рисунок 1. Соотношение требований

## 4.2. Примеры подхода к разработке требований

Приведенные ниже примеры относятся к разработке программного обеспечения для устройств цифрового телевидения.

### Пример 1: Уменьшенное окно для видео

Архитектура системы предусматривает уменьшенное окошко, в котором отображается ТВ канал или записанное видео в то время, как пользователь работает с меню или настройками.

Особенность работы системного инженера состоит в том, что он должен представлять себе всю систему в целом, и продумывать ее поведение в различных ситуациях. Приведем список вопросов, на которые должен ответить системный инженер для того, чтобы понять, что будет отображаться в уменьшенном окошке в разных случаях:

- 1) Во время инициализации системы
- 2) В случае конфликта ресурсов
- 3) При возникновении граничных ситуаций: например, после завершения видео

### Пример 2: Запись

В устройстве предусмотрено два типа записи с телевидения:

1. Запись по заданному временному интервалу
2. Программно-ориентированная запись

Системный инженер должен рассмотреть следующие ситуации:

- Отображение записи по временному интервалу в программно-ориентированной среде
- Сбор и хранение метаданных (жанр программы, актеры, студия и т.д.)
- Остановка или отмена записи
- Включение в запись данных из буфера
- Разрешение конфликтов ресурсов

Требования реализации могут определять такие параметры, как размер буфера записи; возможность начинать запись за несколько минут до начала программы; конкретные опции, предлагаемые пользователю для разрешения конфликта ресурсов, и т.п.

## **5. Заключение**

Хотя описанные методы были опробованы на работе в области цифрового телевидения, нет никаких причин, по которым они не могли бы быть с успехом применены и в других областях, связанных с разработкой программного обеспечения. В этом случае системный инженер оказывается связующим звеном между группой маркетинга, дизайнерами, разработчиками и тестировщиками, что позволяет всем участникам проекта лучше сосредоточиться на своей задаче

На основе опыта, полученного в Санкт-Петербургском центре Моторола, можно сделать вывод о том, что выделение группы системных инженеров способствует лучшей координации и структурированности проекта создания программного обеспечения. Подобная специализация позволяет не только избежать создания неполного представления о системе (ограниченного точкой зрения других участников процесса), но и достигнуть значительной (не менее чем двукратной) экономии ресурсов в процессе определения требований за счет переиспользования ранее созданных требований для других продуктов и накопленного опыта по их разработке и формулировке.

## **6. Используемая литература**

1. Larisa Melikhova, Albert Elcock, Andrey A. Dovzhikov, Georgii Bulatov, Dmitry O. Vavilov, " Reengineering for System Requirements Reuse: Methodology and Use-Case", Proceedings of the 11th Annual IEEE International Symposium on Computer Electronics (ISCE 2007), 20-23 June 2007.
2. Jansma, P.A.; Derro, M.E. If You Want Good Systems Engineers, Sometimes You Have To Grow Your Own! Aerospace Conference, 2007 IEEE. 3-10 March 2007
3. A.P. Sage. Systems Engineering of Computer Based Systems: status and future perspectives. Proceedings of the 1995 International Symposium and Workshop on Systems Engineering of Computer Based Systems. 1995

# Maximizing Intel® Compiler Performance Using Iterative Feedback Directed Optimization

**Artem Chirtsov**  
Intel Corporation  
artem.s.chirtsov@intel.com

**Leonid Brusencov**  
Intel Corporation  
leonid.brusencov@intel.com

**Ilya Cherny**  
Intel Corporation  
ilya.s.cherny@intel.com

**Sergey Grebenkin**  
Intel Corporation  
sergey.s.grebenkin@intel.com

**Sergey Ermolaev**  
Intel Corporation  
sergey.n.ermolaev@intel.com

## Abstract

Iterative search of compiler optimization parameters is a popular method to increase performance of compiled code relatively to default compilation. Most of the existing papers apply optimization parameters to the whole application, while our paper describes results got by the specially modified compiler capable to parameterize each function independently. Six well-known search algorithms were implemented and used to search the best optimization parameters in the space of all possible combinations of parameters values. Standard SPEC CPU2000 benchmarks were compiled using internal version of Intel® Compiler 10.1. Applications were executed on Intel® Core 2™ with Windows\* 2003 OS platforms. Six compiler performance options having the most impact on performance were selected.

As a result we got the maximum performance increase of 11% for 187.facere benchmark and 4% performance increase of total SPEC CPU2000 execution time. Search algorithms comparison shows that most algorithms get very close results varying within 3% from each other. Some algorithms could get their best results after 10-40 iterations while others could get the same results only after 100 iterations which was the limit of a number of iterations.

**Keywords:** iterative, compiler, optimization, feedback directed, parameters, options, performance



# **Максимизация Производительности Компилятора Интел в Режиме Итеративной Оптимизации с Обратной Связью**

**Artem Chirtsov**  
**Intel Corporation**  
artem.s.chirtsov@intel.com

**Leonid Brusencov**  
**Intel Corporation**  
leonid.brusencov@intel.com

**Ilya Cherny**  
**Intel Corporation**  
ilya.s.cherny@intel.com

**Sergey Grebenkin**  
**Intel Corporation**  
sergey.s.grebenkin@intel.com

**Sergey Ermolaev**  
**Intel Corporation**  
sergey.n.ermolaev@intel.com

## **Тезисы**

Очень популярной техникой для улучшения производительности скомпилированного кода относительно стандартных настроек компилятора сегодня стал итеративный перебор параметров оптимизаций компилятора. Большинство подходов используют для параметризации всё приложения целиком, в данной же статье описывается работа модифицированного компилятора, с которым ведётся параллельный поиск наилучшего набора параметров оптимизаций для каждой функции по-отдельности. Для самого поиска мы выбрали и реализовали шесть наиболее известных алгоритмов и сравнили результаты их работы. Для измерения производительности мы использовали стандартный набор тестов SPEC CPU2000, компьютеры на базе процессоров Intel® Core 2™, Microsoft Windows\* 2003 и модифицированный компилятор Intel® Compiler версии 10.1. Мы выбрали шесть опций компилятора, влияющие на производительность получаемого кода наибольшим образом.

В результате мы получили максимальный прирост производительности на отдельном тесте 11%, а на всей сьюте в сумме – 4%. Сравнение алгоритмов показало, что различные алгоритмы добиваются в конечном счёте очень близких результатов, в пределах 3% друг от друга, но некоторым для этого достаточно было только 10-40 итераций, тогда как другие добивались сходного результата ближе к 95 итерациям (мы выставляли ограничение на 100 итераций максимум).

**Keywords:** итеративное компилирование, оптимизация, обратная связь, итеративный поиск, производительность, параметры оптимизаций, опции компилятора..

## 1. Введение

Чтобы добиться максимальной производительности пользовательских приложений, разработчики компиляторов постоянно увеличивают количество применяемых оптимизаций и изменяют эвристики алгоритмов. Однако, большинство оптимизаций не дают универсальной выгоды для всех приложений и всех сред исполнения (операционная система, вычислительное устройство и т.д.). К тому же, достаточно трудно оценить эффект конкретной оптимизации на финальный код из-за сложной зависимости применяемых оптимизаций друг от друга. Подобные проблемы можно избежать, используя подход итеративного компилирования, то есть метод перебора всех возможных вариантов применяемых оптимизаций.

В данный момент чтобы скомпилировать продукт оптимальным образом, что приходится осуществлять вручную, требуется огромное количество времени, нужно знать и помнить большое количество различных параметров компилятора, выборочно их использовать, затем самостоятельно получать результат и сравнивать его с результатами при других наборах параметров. Мало того, требуется быть знакомым со специализированными программами, измеряющими время исполнения блоков кода, чтобы уметь корректно оценивать результат оптимизации.

Однако, если просто автоматизировать этот процесс, подобная система неизбежно оказывается слишком требовательной ко времени поиска из-за необходимости постоянного перекомпилирования и запуска программы. Но есть возможность сужения пространства поиска наборов опций оптимизации, используя специальные алгоритмы поиска.

В нашей работе используется разбиение программы на отдельные функции, но в дальнейшем будет реализовано разбиение и на более мелкие блоки, например на отдельные циклы. Это позволяет получить дополнительный прирост производительности, которого было невозможно добиться для всей программы целиком.

Для сужения пространства поиска были реализованы шесть различных известных алгоритмов поиска и проведен анализ их работы.

Чем может оказаться полезным наш подход? Скомпилированный оптимальным образом код для трудоёмких вычислений или обработки большого количества данных может позволить получить требуемый результат за гораздо меньшее время, или же повысить качество вычисления с помощью более сложного алгоритма, но за то же время. Максимально возможная оптимизация также очень важна для встраиваемых (embedded) систем.

## **2. Связанные работы**

Сегодня итеративные оптимизации - это очень популярный подход в ответ на всё время усложняющиеся архитектуры процессоров. Так, например, в работе [1] демонстрируется, что полный поиск в пространстве параметров оптимизаций может дать заметный (в статье в 2.65 раз на примере задачи перемножения матриц) прирост производительности в сравнении со статическими моделями, а в [2] дополнительно доказывается почему именно такие работы приносят прирост производительности на современных архитектурах.

Появились публикации, в которых объясняется, как данный подход можно использовать на практике, например, для улучшения параметров оптимизации в библиотеках [3] или для лучшего определения статической модели выбора параметров оптимизаций [4].

Существует проект MILEPOST GCC, описанный в [5], в котором применяется подход машинного самообучения. После нескольких недель накопления статистики удалось добиться 11% уменьшения времени исполнения на тестах MiBench на платформах x86 и IA64.

В [6] описывается подход исследования уровней оптимизации компилятора для автоматического их конструирования, чтобы они представляли оптимальный компромисс между несколькими оценивающими функциями,

такими как время исполнения, компилирования, размер кода и другие. Такой компромисс определяется как уровень оптимизации, при котором на всех оценивающих функциях достигается максимум. При этом один уровень доминирует над другим, если он достигает лучшей оценки хотя бы на одной из оценивающих функции, и не хуже на остальных. Для оценки результатов использовалась hyper-volume (HV) метрика, описанная в [7], набор тестов SPEC CPU, процессор Intel Pentium 4 и компилятор GCC. Были получены уровни оптимизации значительно лучшие, чем стандартные -Os, -O1, -O2 и -O3; при оптимизации всех тестов целиком удалось получить до 30% прироста по HV-метрике.

В статье [8] утверждается, что подход, который используется в большинстве итеративных поисков, когда поиск производится на одних и тех же входных данных, демонстрирует лишь потенциал, но не как программа будет работать в реальных условиях. Эта проблема решается подбором большого количества различных входных данных. В самой статье использовался набор из 20 наборов данных для каждого теста из 26 в MiBench сюите. Эксперименты производились на кластере из 16 AMD Athlon 64 3700+ с помощью PathScale EKIPath Compiler 2.3.1, специально настроенного для AMD процессоров. Было получено, что хотя характеристики программы, такие как производительность, или энергопотребление, могут сильно различаться при использовании нескольких оптимизаций при компилировании, чаще всего возможно найти такой компромиссный оптимальный набор параметров оптимизаций, который бы работал на всех входных данных одинаково. Причём отставание от скомпилированной программы с оптимальными оптимизациями для конкретного набора данных не превышает всего лишь 5%.

В области поиска оптимального набора оптимизаций существует направление, которое называют мультиверсионностью или клонированием [9, 10, 11]. Оно основано на создании нескольких скомпилированных вариантов функций, специализированных на определённые

параметры вызова, что позволяет оптимизировать их более агрессивно. При этом ветвление происходит либо статически, либо динамически, либо используя накопленную статистику (например, в [12] в компилятор GCC внедряется машинное самообучение). Такой подход используется как некоторое расширение для компилятора, но не покрывает все случаи различных входных данных в программу.

В работе [13] описывается подход сходный работе JIT (just-in-time) компиляторов. Принцип работы заключается в интерпретации кода для динамического выявления наиболее часто исполняемых мест программы, с последующей их оптимизацией и трансляцией в машинный код. Причём оптимизация ведётся с учётом известных входных данных, то есть часть переменных используются как константы, что добавляет эффективности.

В работе [14] динамически вставляются инструкции упреждающего чтения из памяти, основываясь на счётчиках процессора. В работе [15] используются высокоуровневые оптимизации кода во время исполнения либо в параллельном процессе, либо вообще на удалённой машине. Главная особенность данной работы – она позволяет итеративно искать и применять динамические оптимизации. Пользователю предлагается специальный язык, в котором можно указывать множество эвристик, которые будут применены к определенным участкам программы; затем полученные варианты кода динамически исследуются при запуске, и выбирается лучшее решение. Кроме этого система позволяет генерировать новые варианты кода, основываясь на опыте оптимизации других фрагментов кода.

В работе [16] в исследуемой программе выделяются фазы исполнения со сходной производительностью, существование которых показывается в [17, 18]. Таким образом, на каждой фазе исполнения можно использовать различные оптимизации, то есть в одном только запуске попробовать сразу несколько решений.

### 3. Описание реализованных в проекте алгоритмов поиска

Рассмотрим выбранные нами алгоритмы поиска максимума производительности в пространстве всевозможных комбинаций параметров оптимизаций.

#### 3.1. Алгоритм полного перебора

Алгоритм полного перебора (Exhaustive Search algorithm) использует каждый из допустимых наборов опций компиляции (то есть строит полное пространство решений), в связи с этим найденный минимум целевой функции всегда является глобальным. Однако вычислительная сложность данного алгоритма равна  $O(m^n)$ , где  $n$  – число опций компиляции,  $m$  – число возможных значений опций. Такая сложность считается неприемлемой, поскольку время поиска растёт экспоненциально с ростом числа опций [19], поэтому часто на практике предпочтение отдаётся различным модификациям. В данной работе был использован алгоритм полного перебора с приоритетом: каждой опции и её значению на основе экспертной оценки был приписан некоторый вес, в зависимости от него в процессе формирования очередного тестового набора те или иные опции получали более высокий приоритет. В результате при завершении итераций задолго до окончания полного перебора уже будут опробованы наиболее значимые комбинации опций и их параметров.

#### 3.2. Алгоритм группового исключения

Идея алгоритма группового исключения (Batch Elimination Algorithm) заключается в том, чтобы определить опции, которые способствуют увеличению времени исполнения тестируемой программы, и не использовать их в дальнейшем поиске оптимального набора опций. Реализация данного алгоритма достигает хорошей производительности, если влияние опций друг на друга незначительно.

Для определения эффекта использования какой-либо опции используется Относительная Процентная Доля Улучшения (Relative Improvement Percentage, далее RIP), которая показывает относительную разницу времени исполнения тестируемой программы, скомпилированной в двух вариантах: с выключенной опцией и со всеми включёнными опциями. Если полученная величина отрицательна, данный алгоритм исключает из набора исследуемую опцию, получая, таким образом, оптимальный набор. Вычислительная сложность алгоритма составляет  $O(n)$  [20].

### 3.3. Алгоритм итеративного исключения

В отличие от алгоритма группового исключения, алгоритм итеративного исключения (Iterative Elimination Algorithm) последовательно исключает из стартового набора опции, которые увеличивают время исполнения тестируемой программы. Стартовый набор для данного алгоритма ничем не отличается от стартового набора для алгоритма группового исключения. Обозначим через  $B_i$  набор опций  $B$ , из которого исключили опцию  $F_i$ , тогда после того как для всех наборов  $B_i$  значения  $RIP$  посчитаны, алгоритм создаёт новый стартовый набор  $B^{(1)}$ , исключая из него опцию, имеющую  $RIP$  с наименьшим отрицательным значением. После чего снова считают  $RIP$  для всех наборов  $B_i^{(1)}$ . Итеративный процесс продолжается до тех пор, пока не будут исчерпаны все опции с отрицательным значением  $RIP$ . Вычислительная сложность данного алгоритма составляет  $O(n^2)$  [20].

### 3.4. Алгоритм комбинированного исключения

Алгоритм комбинированного исключения (Combined Elimination Algorithm) совмещает в себе идеи алгоритмов группового и итеративного исключения. Если опции

компиляции слабо влияют друг на друга, данный алгоритм исключает опции с отрицательным значением *RIP*, как алгоритм группового исключения, в противном случае он исключает их в течение итеративного процесса, как алгоритм итеративного исключения. Сложность данного алгоритма равна  $O(n^2)$  [21].

### 3.5. Генетический алгоритм

Генетический алгоритм (Genetic Algorithm) позволяет найти удовлетворительное решение к аналитически неразрешимым проблемам через последовательный подбор и комбинирование искоемых параметров с использованием механизмов, напоминающих биологическую эволюцию, в процессе которой выживают наиболее приспособленные особи, а наименее приспособленные погибают.

Алгоритм начинает свою работу с формирования начальной популяции – конечного набора допустимых решений задачи. Эти решения выбираются случайным образом. На каждом шаге эволюции с помощью вероятностного оператора селекции выбираются два решения – родители. Оператор скрещивания по выбранным решениям строит новое, которое подвергается небольшим случайным модификациям – мутациям. Затем решение добавляется в популяцию, а решение с наименьшим значением целевой функции удаляется из популяции [22].

### 3.6. Алгоритм статистического выбора

Алгоритм статистического выбора (Statistical Selection Algorithm) использует статистику вывода (Inferential Statistics). Вводятся экспериментальная и начальная гипотезы, которые противоречат друг другу. Для проверки гипотез определяют контрольную и экспериментальную группы для каждой из опций, но поскольку для их проверки требуется значительное число запусков тестируемой программы с различным набором опций, создаётся только одна контрольная и одна

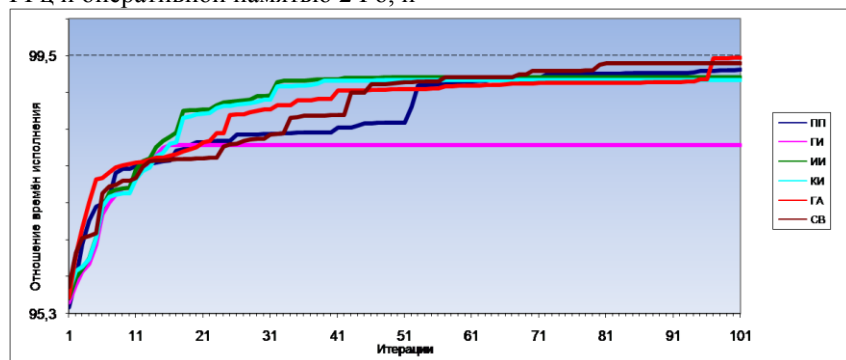


экспериментальная группа, в которой значения опций распределены случайным образом с использованием ортогональных массивов, к которым применяется тест Манна-Уитни для расчёта влияния той или иной опции на время программы.

Чтобы определить, влияет ли некоторая опция на время исполнения тестируемой программы, информация о времени исполнения распределяется на две группы – экспериментальную и контрольную, затем, используя информацию экспериментальной группы, вычисляется тестовая статистика, и на её основе при помощи вероятностной функции определяется, попадает ли опция с данным значением в оптимальный набор [23].

#### 4. Методы тестирования

Для тестирования был выбран набор стандартных тестов производительности процессора и памяти SPEC CPU2000 с reference данными, компилятор Intel® Compiler 10.1, компьютеры с процессорами Intel® Core 2™ с частотой 2.40 ГГц и оперативной памятью 2 Гб, и



**Рисунок 9. График зависимости от итерации суммарного времени исполнения функций всех тестов для каждого из алгоритмов. Показано процентное соотношение минимального времени к текущему.**

набор из шести недокументированных опций компилятора, отвечающих за векторизацию (vectorize), слияние циклов (fusion), разбиение циклов (distribution), разворачивание циклов (unroll, unroll and jam), разбиение на блоки (blocking) и инструкции упреждающего чтения из памяти (prefetch). Для измерения времени работы отдельных функций программы использовался встроенный в компилятор профилировщик, который инструментировал код.

Каждый алгоритм запускался для каждого отдельного теста с ограничением в 100 на максимальное количество итераций. Затем полученные данные анализировались и строились сравнительные графики.

Тестирование проводилось с помощью специально разработанного приложения, производящего автоматически построение и запуск тестов с сохранением данных о результатах в собственную базу данных. Для сравнения было взято шесть описанных выше алгоритмов: алгоритм полного перебора с приоритетом (ПП), алгоритм группового исключения (ГИ), алгоритм итеративного исключения (ИИ), алгоритм комбинированного исключения (КИ), генетический алгоритм (ГА), алгоритм статистического выбора (СВ).

## **5. Сравнительный анализ**

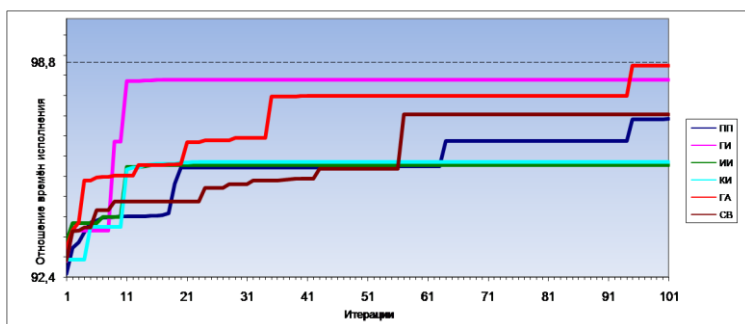
Рассмотрим график зависимости суммарного времени исполнения всех функций всех тестов от номера итерации (Рисунок 9). Было исследовано 164 функций (всего функций 215), входящих в состав различных тестов SPEC CPU2000 (всего 26 тестов). Анализировались только те функции, минимальное время исполнения которых составляло более 1% от минимального времени исполнения данного приложения для всех алгоритмов, чтобы уменьшить уровень шума.

График показывает процентное соотношение суммы минимальных времён исполнения всех функций к сумме времён всех функций рекордных для данной итерации данного алгоритма. Таким образом, график демонстрирует насколько быстро каждый из алгоритмов поиска приближается к

максимуму производительности в пространстве всех комбинаций параметров оптимизаций.

Как видно из графика трудно выделить явного лидера, поскольку результаты лидеров отличаются лишь на 0,4%. Наилучшие результаты показали генетический алгоритм, алгоритм статистического выбора и полного перебора (относительное уменьшение времени составило 3,90%, 3,87% и 3,65% соответственно).

Очень близкие друг к другу результаты разных алгоритмов объясняются усреднением по большому количеству функций (164). При рассмотрении отдельных тестов (пример ниже) или даже отдельных функций разные алгоритмы достигают разных максимумов производительности, а также приближаются к ним существенно по-разному.



**Рисунок 10. График зависимости от итерации суммарного времени исполнения всех функций теста CPU2000 SPEC 186.crafty для каждого из алгоритмов.**

Если анализировать итеративный процесс, можно заметить, что алгоритмы статистического выбора и полного перебора с приоритетом приближаются к оптимальному результату позже всех. Для первого – это связано с фазой пробных запусков, в процессе которых идёт накопление информации, для второго – с фиксированным порядком перебора не являющимся наиболее оптимальным для широкого круга функций. Таким образом, эти алгоритмы будут давать тем более точный результат, чем больше запусков будет произведено. Алгоритмы итеративного и комбинированного исключения раньше всех достигли значений близких к лучшим найденным, что при сильно ограниченном количестве итераций позволяет им показывать наилучшие результаты. Алгоритм группового исключения показал наихудший результат и остановил работу уже на 16 итерации, однако в некоторых случаях он может становиться лидером.

Например, рассмотрим такой же график для приложения 186.crafty (Рисунок 2), которое представляет собой шахматную программу. На графике видно, что наилучший результат показал генетический алгоритм, однако значительно раньше вышел в лидеры и долго им оставался (86 итераций) алгоритм группового исключения, который на общем графике занимал последнюю позицию; в то же время алгоритм итеративного исключения уступил лидеру почти 2,95%. Полученный результат обусловлен очень слабым влиянием опций друг на друга в данном приложении, что в общем случае выражается не так явно.

Наибольшее относительное уменьшение времени в 11.3% было получено алгоритмом итеративного исключения для приложения 187.facesec – исходный код для программы распознавания лиц, который был наиболее восприимчив к выбранным опциям оптимизаций. Почти все тесты (25 из 26) показали прирост производительности более 1%. Распределение прироста производительности по тестам показано на графике (Рисунок 11).

## **6. Заключение**

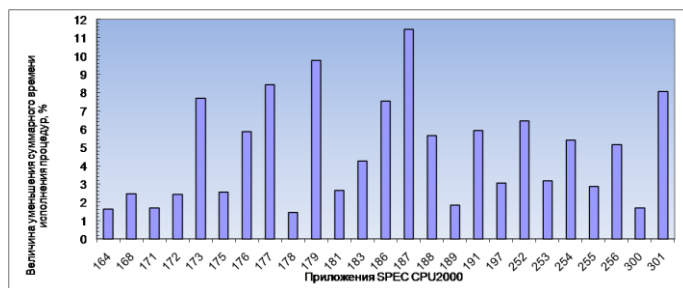
Проведенные эксперименты по влиянию шести опций управления оптимизациями компилятора на производительность показывают жизнеспособность данного подхода для широкого набора приложений. Для отдельных тестов прирост производительности в 8-11% достигается уже на

10 итерациях построения и запуска при использовании алгоритмов исключения. В среднем необходимо около 40 итераций для улучшения суммарной производительности на 3-4%.

В дальнейших работах будет в десятки раз расширен набор опций компилятора, а также опробовано применение самообучающейся экспертной системы, способной на основе свойств приложений существенно сократить число необходимых итераций вплоть до одной - статической компиляции.

## **7. Список литературы**

[1] F. Bodin, T. Kisuki, P. Knijnenburg, M. O'Boyle, and E. Rohou, "Iterative compilation in a non-linear optimization space", In Proc. ACM Workshop on Profile



**Рисунок 11. График прироста производительности для отдельных тестов SPEC CPU2000**

and Feedback Directed Compilation, 1998, Organized in conjunction with PACT98.

[2] K. Cooper, D. Subramanian, and L. Torczon, “Adaptive optimizing compilers for the 21<sup>st</sup> century”, J. of Supercomputing, 32(1), 2002.

[3] J. Bilmes, K. Asanovic, C. Chin, and J. Demmel, “Optimizing matrix multiply using PHiPAC: A portable, high- performance, ANSI C coding methodology”, In Proc. ICS, pages 340-347, 1997.

[4] M. Stephenson and S. Amarasinghe, “Prediction unroll factor using supervised classification”, In ERRR/ACM International Symposium on Code Generation and Optimization (CGO 2005), ERRR Computer Society, 2005.

[5] Yom-Toy, J. Thomson, O. Temam, A. Zaks, H. Leather, C. Miranda, M. Namolaru, E. Bonilla, Saclay, B. Mendelson, C. Williams, Haifa, M. O’Boyle, P. Barnard, E. Ashton, E. Courtois, F. Bodin “MILEPOST GCC: machine learning based research compiler”, ARC, International, UK, CAPS Enterprise, France, 2007.

[6] K. Hoste, L. Eeckhout, “COLE: Compiler Optimization Level Exploration”, ELIS Department, Ghent University, Sing-Pietersnieuwstraat 41, B-9000 Gent, Belgium, 2008.

[7] K. Deb, “Multi-Objective Optimization using Evolutionary Algorithms”, Wiley, 2001.

[8] G. Fursin, J. Cavazos, M. O’Boyle, and O. Temam, “MiDataSets: Creating the Conditions for a More Realistic Evaluation of Iterative Optimization”, ALCHEMY Group, INRIA Futurs and LRI, Paris-Sud University, France, 2007.

- [9] M. Byler, M. Wolfe, J.R.B. Davies, C. Huson, and B. Leasure, "Multiple version loops." In ICPP, 1987, pages 312-318, 2005.
- [10] K. D. Cooper, M. W. Hall, and K. Kennedy, "Procedure cloning", In Proceedings of the 1992 IEEE International Conference on Computer Language, pages 99-105, 1992.
- [11] P. Diniz and M. Rinard. "Dynamic feedback: An effective technique for adaptive computing", In Proc. PLDI, pages 71-84, 1997.
- [12] G. Fursin, C. Miranda, S. Pop, A. Cohen, O. Temam, "Practical Run-time Adaptation with Procedure Cloning to Enable Continuous Collective Compilation", Alchemy group, INRIA Futurs and LRI, Paris-Sud 11 University, Orsay, France, 2007.
- [13] V. Bala, E. Duesterwald, and S. Banerjia, "Dynamo: A transparent dynamic optimization system", In ACM SIGPLAN Notices, 2000.
- [14] R. H. Saavedra and D. Park, "Improving the effectiveness of software prefetching with adaptive execution", In Conference on Parallel Architectures and Compilation Techniques (PACT'96), 1996.
- [15] M. Voss and R. Eigemann, "High-level adaptive program optimization with adapt", In Proceedings of the Symposium on Principles and practices of parallel programming, 2001.
- [16] G. Fursin, A. Cohen, M. O'Boyle, and O. Temam, "A Practical Method For Quickly Evaluating Program Optimizations", Institute for Computing Systems Architecture, University of Edinburgh, UK, 2005.
- [17] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior", In 10<sup>th</sup> International Conference on Architectural Support for Programming Languages and Operating Systems, 2002.
- [18] J. Lau, S. Schoenmackers, and B. Calder, "Transition phase classification and prediction", In International Symposium on High Performance Computer Architecture, 2005.
- [19] Z. Pan, R. Eignmann, "Fast and Effective Orchestration of Compiler Optimizations for Automatic Performance Tuning.", Proceedings of the International Symposium on Code Generation and Optimization, 2006.
- [20] S. Triantafyllis, M.J. Bridges, E. Raman, G. Ottoni and D. August, "A Framework for Unrestricted Whole-Program Optimization.", Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation, 2006.
- [21] Z. Pan, R. Eignmann, "Fast, Automatic, Procedure-Level Performance Tuning.", Proceedings of the 15th International Conference on Parallel Architecture and Compilation Techniques, 2006.

- [22] H. Feltl, “Ein Genetischer Algorithmus fuer das Generalized Assignment Problem”, Diplomarbeit, 2003.
- [23] M. Haneda, P. Knijnenburg, H. Wijshoff “Automatic Selection of Compiler Options Using Non-Parametric Inferential Statistics.”, Proceedings of the 14th International Conference on Parallel Architecture and Compilation Techniques, 2005.



# **Intel® Concurrent Collections for C++ - a model for parallel programming**

**Nikolay Kurtov**  
**Intel Corporation**  
**email:**

**nikolay.kurtov@intel.com**

**Ilya Cherny**  
**Intel Corporation**  
**email:**

**ilya.s.cherny@intel.com**

**William Youngs**  
**Intel Corporation**  
**email:**

**william.youngs@intel.com**

**Kath Knobe**  
**Intel Corporation**  
**email:**

**kath.knobe@intel.com**

**Geoff Lowney**  
**Intel Corporation**  
**email:**

**geoff.lowney@intel.com**

**Shin Lee**  
**Intel Corporation**  
**email:**

**shin.lee@intel.com**

**Stephen Rose**  
**Intel Corporation**  
**email:**

**stephan.rose@intel.com**

**Leo Treggiari**  
**Intel Corporation**  
**email:**

**leo.treggiari@intel.com**

**Judy Ward**  
**Intel Corporation**  
**email:**

**judy.ward@intel.com**

## **Abstract**

There is a challenge to grow the community of developers who can successfully exploit multi-core because application developers, who are experts in their domains, are not necessarily parallel or performance experts. Application providers have limited resources to develop and maintain multiple target-specific variants of their source code. Parallel programming is just too error-prone and time-consuming for wide scale adoption

Intel® Concurrent Collections is a simple yet powerful parallel programming model which separates the expression of all potential parallelism in an application both from the serial computations and also from all the target-specific details. This model raises the level of a programming language just enough to avoid typical parallelization issues and requires domain experts to define a semantically correct algorithm only.

A program in this model consists of an abstract parallel algorithm definition, high level primitive operations and data

structures implemented in a serial language. All target-specific issues are solved by the runtime system. This is a very general approach of defining a parallel algorithm and it makes easy expressing any kind of parallelism. Intel® Concurrent Collections applications are not target-specific and do not have to be rewritten when they are ported to another platform.

Intel® Concurrent Collections for C++ is implemented as a C++ library and published on [whatif.intel.com](http://whatif.intel.com) site (<http://softwarecommunity.intel.com/articles/eng/3862.htm>).

Implementation includes a translator from textual abstract parallel algorithm definition to classes declaration.

Two standard performance benchmarks were modified to deploy Intel® Concurrent Collections for C++. Performance analysis demonstrated viability of the model and its implementation showing excellent scalability in some cases. Also some future areas for improvement of the implementation were identified.

**Keywords:** Parallel programming; C++ library; multi-core; multi-threading.

## **Модель параллельного программирования Intel® Concurrent Collections для C++**

<b>Nikolay Kurtov</b> Intel Corporation email: <b>nikolay.kurtov@intel.com</b>	<b>Ilya Cherny</b> Intel Corporation email: <b>ilya.s.cherny@intel.com</b>	<b>William Youngs</b> Intel Corporation email: <b>william.youngs@intel.com</b>
---	---	---

<b>Kath Knobe</b> Intel Corporation email: <b>kath.knobe@intel.com</b>	<b>Geoff Lowney</b> Intel Corporation email: <b>geoff.lowney@intel.com</b>	<b>Shin Lee</b> Intel Corporation email: <b>shin.lee@intel.com</b>
---	---	---

<b>Stephen Rose</b> Intel Corporation email: <b>stephan.rose@intel.com</b>	<b>Leo Treggiari</b> Intel Corporation email: <b>leo.treggiari@intel.com</b>	<b>Judy Ward</b> Intel Corporation email: <b>judy.ward@intel.com</b>
---	---	---

### **Тезисы**

Интересна задача увеличения сообщества программистов, успешно использующих возможности многоядерных архитектур. Разработчики приложений ограничены в ресурсах на разработку и поддержание различных платформенно-зависимых вариантов исходных кодов. Параллельное программирование слишком сложно и подвержено ошибкам, что замедляет его широкое распространение.

Intel® Concurrent Collections – простая, но очень мощная модель параллельного программирования, отделяющая весь потенциальный параллелизм в приложении и от последовательных вычислений, и от платформенно-зависимых деталей. Эта модель поднимает уровень языка программирования ровно настолько, чтобы избежать типичных проблем параллелизации, и позволяет экспертам предметной области лишь определять семантически корректный алгоритм.

Программа, написанная в модели Intel® Concurrent Collections, состоит из абстрактного определения параллельного алгоритма, высокоуровневых операций и структур данных, реализованных в последовательном языке. Все платформенно-зависимые вопросы решаются системой исполнения. Это очень общий подход к определению параллельного алгоритма, делающий лёгким выражение любого типа параллелизма. Приложения в этой модели не являются платформенно-зависимыми и не нуждаются в переписывании при переносе на другую платформу.

Intel® Concurrent Collections для C++ реализована как C++ библиотека и опубликована на сайте [whatif.intel.com](http://whatif.intel.com) (<http://softwarecommunity.intel.com/articles/eng/3862.htm>).

| Реализация включает в себя транслятор из текстового абстрактного описания алгоритма в определение классов.

Два приложения для анализа производительности системы были реализованы с помощью данной модели и подтвердили её жизнеспособность, в некоторых случаях продемонстрировали хорошую масштабируемость, а также определили области для дальнейшего развития модели.

**Keywords:** Параллельное программирование; библиотека  
| C++; многоядерность; многопоточность.

## 1. Вступление

Многоядерные архитектуры распространяются всё шире, а для полного использования предоставляемых возможностей всё важнее становится параллельное программирование. Само по себе параллельное программирование очень сложно, подвержено ошибкам и зависит от лежащей в основе архитектуры. Сейчас очень интересна задача увеличения сообщества программистов, успешно использующих возможности многоядерных архитектур. Одним из способов решения данной задачи является разработка средств, позволяющих избежать низкоуровневого программирования многопоточности.

Одним из таких средств является проект Intel® Concurrent Collections for C/C++. Это очень простая и мощная модель параллельного программирования, отделяющая выражение всего потенциального параллелизма в приложении и от последовательных вычислений, и от архитектурно-зависимых деталей.

## 2. Модель Intel® Concurrent Collections

В чём основная сложность параллельного программирования? Разработчик должен заботиться о выражении параллелизма, что часто приводит к тому, что разработчик уделяет много внимания не самому приложению, а работе с параллелизмом.

Другая причина, затрудняющая параллельное программирование, – оно встроено в последовательные языки программирования, что влечет за собой явное упорядочивание операций и перезапись данных.

Intel® Concurrent Collections поднимает уровень языка ровно настолько, чтобы избежать упомянутых проблем. Вводится не расширение языка, а интерфейс для описания задачи. Это позволяет решать задачу специалисту предметной области, который не является специалистом в области параллельного программирования. Таким образом, от разработчика требуется написать семантически корректную последовательную программу в рамках накладываемых Intel® Concurrent Collections ограничений. С него снимаются многие низкоуровневые вопросы:

- оптимизация под конкретную архитектуру
- написание платформенно-зависимого кода для работы с потоками
- балансировка загрузки потоков
- распределение задач между процессорами

Решением этих задач может заниматься другой человек, специалист по оптимизации, незнакомый с предметной областью, или статический анализ или анализ времени исполнения.

## 3. Intel® Concurrent Collections и другие модели параллельного программирования

Подход Intel® Concurrent Collections значительно отличается от других существующих моделей параллельного программирования.

При использовании Cilk[1] программист явным образом указывает вызовы функций, которые могут исполняться параллельно, а также о синхронизации при помощи барьеров, но при этом текст программы при удалении из него ключевых слов Cilk является корректной C/C++ программой.

OpenMP[2] очень хорошо подходит для задач, обладающих параллелизмом по данным. Базовые возможности OpenMP очень просты в освоении, что и делает эту модель очень популярной. Использование OpenMP требует поддержки компилятора.

Intel® TBB[3] предлагает высокоуровневую платформеннонезависимую абстракцию для параллельного программирования, а также множество базовых параллельных алгоритмов, которые могут значительно сократить объём кода и увеличить его качество и надёжность. Использование Intel® TBB требует хороших навыков применения шаблонов языка C++.

Во всех вышеперечисленных моделях программисте явно заботится о выражении параллелизма и имеет дело с императивным описанием алгоритма. Такой подход позволяет иметь больший контроль над производительностью программы, но заставляет разработчика думать о многих низкоуровневых вопросах.

Подход Intel® Concurrent Collections отличается тем, что предлагаемое естественное функциональное представление алгоритма требует лишь явного указания зависимостей в алгоритме, что делает возможным выявление всего потенциального параллелизма алгоритма. Также такое описание алгоритма является значительно более общим, что позволяет эффективно выражать любой тип параллелизма.

С другой стороны, скрывая все низкоуровневые вопросы параллелизма от разработчика, Intel® Concurrent Collections также лишает его возможности значительно влиять на производительность программы.

#### **4. Основные понятия**

Основные понятия будут рассмотрены на примере простого приложения Blackscholes, решающего дифференциальное уравнение Блэка-Шоулза для множества наборов входных параметров. Данная программа представлена в наборе приложений для тестирования производительности системы PARSEC[4].

Способ описания задачи в модели Intel® Concurrent Collections представляет собой взгляд разработчика на решаемую задачу, который дополнен ровно настолько, чтобы сделать возможным исполнение данного представления.

В описании алгоритма разработчик отвечает на следующие вопросы:

- Какие выделены высокоуровневые операции?
- Какие данные используются?
- Какие присутствуют отношения производителя/потребителя?
- Какие данные являются входными, какие выходными?

Графическое представление алгоритма приложения Blackscholes представлено на рисунке 1.



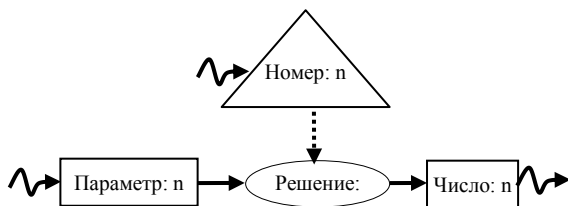
**Рисунок 1. Графическое представление алгоритма Blackscholes**

Таким образом выделены два типа сущностей: высокоуровневые операции, называемые шаг (Step), и элементы данных (Item). В примере Blackscholes шаг называется «Решение», элементы данных называются «Параметры» и «Ответ». Прямые стрелки обозначают отношения производителя и потребителя, а фигурными стрелками обозначены входные и выходные данные.

Для уточнения этого описания разработчику требуется ответить также на следующие вопросы:

- Как различаются экземпляры шагов и элементы данных?
- Каковы наборы шагов и элементов данных?
- Кем определяются эти наборы?

Для ответа на эти вопросы к экземплярам шагов и элементов



**Рисунок 2. Графическое описание алгоритма Blackscholes**

данных добавляются уникальные идентификаторы (Tag), имеющие произвольное число атрибутов произвольного типа.

Набор однотипных элементов данных образует пространство элементов данных (Item Space), набор шагов образует пространство шагов (Step Space) и набор идентификаторов образует пространство идентификаторов (Tag Space).

На рисунке 2 изображен алгоритм приложения Blackscholes в графическом представлении Intel® Concurrent Collections. Пространства шагов обозначаются овалом, идентификаторов – треугольником, а пространства элементов данных обозначаются прямоугольниками.

Также вводится понятие отношения между сущностями. Сущности могут находиться в отношении подписывания (Prescriptive relations), что на графическом представлении обозначается пунктирной стрелкой, или в отношении производителя и потребителя (Producer and consumer relations) – сплошная стрелка.

В отношении подписывания могут находиться только пространство идентификаторов и пространство шагов. При этом для каждого идентификатора из первого пространства будет создан и исполнен шаг второго пространства, помеченный этим идентификатором. При этом нет ограничений на число пространств шагов, подписанных одним и тем же пространством идентификаторов.

В отношении потребителя состоят пространство элементов данных и пространство шагов, при этом шаг может использовать данные из указанного пространства элементов данных.

Отношение производителя – отношение между пространством шагов и пространством элементов данных или идентификаторов. Это означает, что шаг может производить элементы данных указанного типа или идентификаторы, что делает доступными для исполнения новые шаги.

Как эта модель работает для примера Blackscholes? До начала вычислений программист добавляет в пространство элементов данных все необходимые наборы параметров и для каждого из них создаёт соответствующий идентификатор. После запуска вычислений для каждого созданного идентификатора выполняется высокоуровневая операция, использующая данные из пространства «Параметры» и производящая элементы данных «Ответ». После выполнения всех вычислений результирующие данные становятся доступны программисту для дальнейшей работы с ними.

В таком описании алгоритма разработчику нигде не приходится заботиться о параллелизме приложения, тем не менее, он должен соблюдать некоторые естественные ограничения. Предполагается, что шаги выполняются атомарно, шаги могут взаимодействовать с глобальными данными только путём добавления и получения данных из пространств элементов. Запрещается модифицировать глобальные данные напрямую, это может повлиять на результат выполнения других шагов.



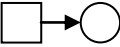
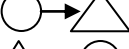
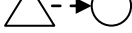
Таким образом, вводится достаточно простая модель параллельного программирования, освобождающая специалиста предметной области от низкоуровневых вопросов параллельного программирования. Модель является универсальной и может быть применена практически к любому языку и архитектуре: как к



архитектуре с общей памятью, так и к архитектуре с распределённой памятью.

## **5. Текущая реализация для C++**

На данный момент имеется реализация модели Intel® Concurrent Collections для C++, использующая библиотеку для параллельного программирования Intel® Threading Building Blocks, выполненная в виде шаблонной библиотеки. Высокоуровневое описание алгоритма производится не в графическом, а в текстовом виде. Правила текстового описания очень просты, на рисунке 3 приведены текстовые эквиваленты графическим обозначениям.

Элемент данных		<code>[]</code>
Шаг		<code>()</code>
Идентификатор		<code>&lt;&gt;</code>
Потребитель		<code>[] -&gt; ()</code>
Производитель		<code>() -&gt; []</code>
		<code>() -&gt; &lt;&gt;</code>
Подписка		<code>&lt;&gt; :: ()</code>

**Рисунок 3. Соответствие между графическим и текстовым представлениями**

Для Blackscholes текстовое описание алгоритма представлено на рисунке 4.

```
//Объявления
[Data* parameters];
[double results];
<number: int n>;
//Отношения подписки
<number> :: (Solve);
//Отношения производителя/потребителя
env -> <number>;
env -> [parameters];
[parameters] -> (Solve);
(Solve) -> [results] -> env;
```

**Рисунок 4. Текстовое представление алгоритма Blackscholes**

Входные и выходные данные обозначены как производимые или потребляемые окружением. В данном случае окружение – это C++ программа, производящая начальную подготовку данных и их обработку после завершения исполнения описанного графом алгоритма. В описании элементов данных также необходимо указать тип данных, которые содержатся в элементах. Это может быть как примитивный тип языка C++, так и тип данных, определённый

пользователем. В определении идентификаторов необходимо указать их атрибуты. В текущей реализации тип атрибутов может быть только примитивный целочисленный тип `int`, а их число ограничено пятью. Также в текстовом описании разрешены однострочные комментарии, начало которых обозначается двумя косыми чертами.

Определение класса графа генерируется с помощью транслятора из текстового представления алгоритма. Транслятор производит заголовочный C++ файл, который включается разработчиком в программу. Также транслятор генерирует текстовый файл с шаблонами определения шагов, которые могут сильно помочь в работе с интерфейсом Intel® Concurrent Collections.

Шаги описываются как обычные C++ функции, но имеют определённую сигнатуру, пример шага «Решение» приложения Blackscholes представлен на рисунке 5.

```
StepReturnValue_t Solve(  
    blackscholes_graph_t& graph,  
    const Tag_t& step_tag) {  
    Data* parameters =  
        graph.parameters.Get(step_tag);  
    double result =  
        SolveBlkSchlsEquation(parameters);  
    graph.result.Put(step_tag, result);  
    return CNC_Success;  
}
```

**Рисунок 5. Описание шага «Решение» в приложении Blackscholes**

Возвращаемое значение – это одна из двух специальных констант: `CNC_Success` и `CNC_Failure`. Первое сигнализирует об успешном завершении вычислений, тогда как при возвращении `CNC_Failure` шаг не считается завершенным и его исполнение будет повторено позже.

Аргументами функции шага являются граф и идентификатор исполняемого шага. Граф предоставляет доступ к необходимым пространствам элементов данных с помощью методов `Get` и `Put`, первый служит для получения элемента данных по его идентификатору, второй – для добавления элемента данных с указанным идентификатором. В пространство идентификаторов можно лишь добавлять новые экземпляры с помощью метода `Put`.

Для запуска алгоритма требуется создать объект графа данного алгоритма, инициализировать пространства элементов данных и идентификаторов входными данными и вызвать у графа метод `run`.

## **6. Перезапуск шагов**

Порядок исполнения шагов неопределён, поэтому могут возникать ситуации, когда запущенный на исполнение шаг требует ещё не доступные данные. В этом случае продолжение исполнения шага невозможно, он прерывается и помещается в локальную очередь ожидания этого элемента данных. Это означает, что в момент, когда элемент данных станет доступен, все ожидавшие его шаги будут перезапущены.

Такой механизм позволяет значительно упростить работу программиста, но и накладывает дополнительные ограничения на определение шага:

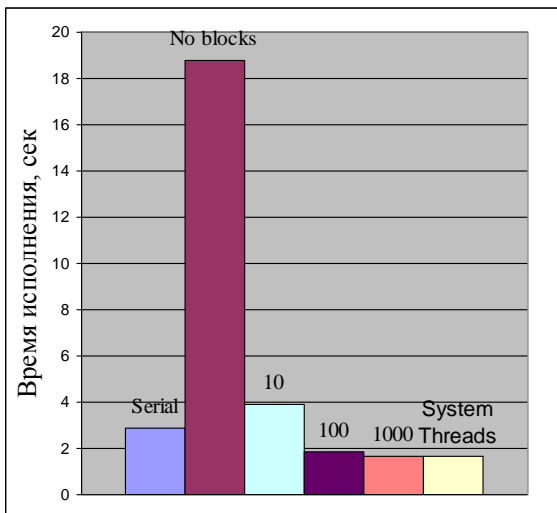
- Все методы `Get` должны вызываться раньше любых методов `Put`
- До выполнения всех методов `Get` нельзя выделять или освобождать память
- Шаги не должны иметь побочных эффектов

Не соблюдение этих ограничений может привести к утечке памяти и непредсказуемой работе программы.

Механизм перезапуска шагов позволяет запускать доступные шаги в произвольном порядке, не влияя на корректность программы.

## **7. Использование блочных алгоритмов**

Несмотря на простоту в применении модели, часто можно получить значительный выигрыш в производительности, выполняя операции не для отдельных объектов, а для блоков объектов. Особенно это становится важно, когда вычисления для одного объекта малы, менее 5000 инструкций. В этом случае накладные расходы на создание и извлечение элементов данных и идентификаторов из их пространств, создание и запуск шагов становятся очень весомыми.



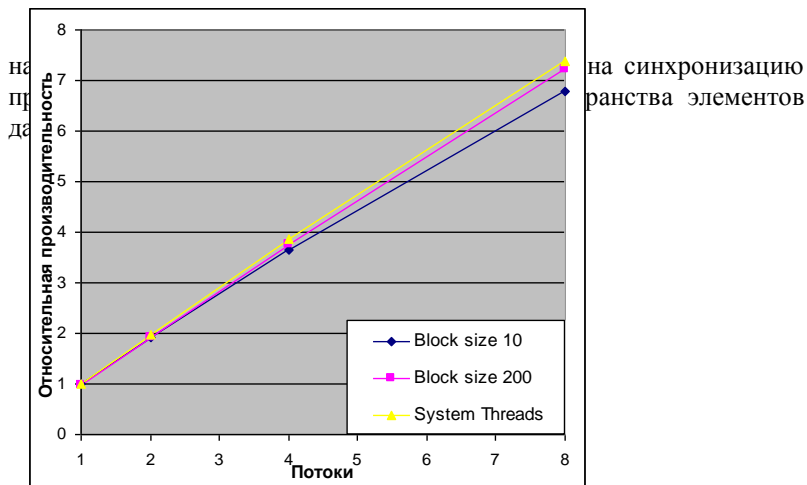
**Рисунок 6. Сравнение разных вариантов алгоритма Blackscholes**

В приложении Blackscholes решение дифференциального уравнения для одного набора входных параметров занимает менее 500 инструкций процессора. На рисунке 6 представлено сравнение времени исполнения следующих вариантов алгоритма Blackscholes для решения задачи на восьми потоках для 5000000 различных наборов параметров:

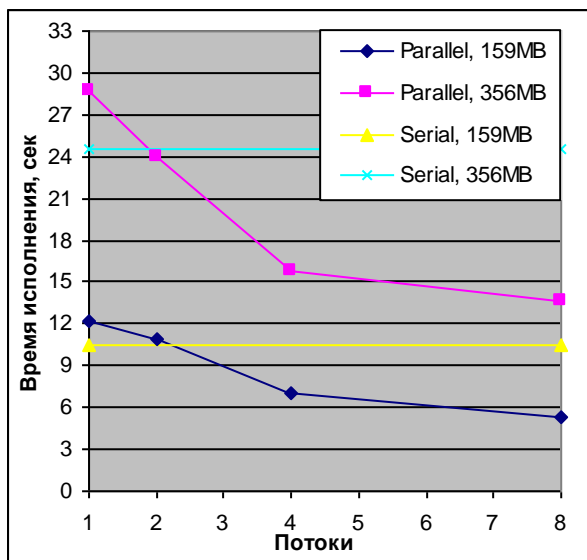
- Последовательная программа
- Один набор параметров за шаг
- 10 наборов параметров за шаг
- 100 наборов параметров за шаг
- 1000 наборов параметров за шаг
- Явное использование системных потоков

При явном использовании программистом системных потоков путем системных вызовов ОС для распараллеливания Blackscholes, разделении объема работ поровну между всеми потоками, может быть достигнута производительность на 2% лучше, чем при использовании Intel® Concurrent Collections.

На рисунке 7 представлены графики производительности разных вариантов алгоритма Blackscholes по отношению к версии, использующей системные потоки, исполняющейся на одном потоке. Масштабируемость версии алгоритма с размером блоков 200 меньше



**Рисунок 7. Масштабируемость разных вариантов алгоритма Blackscholes**



**Рисунок 8. Время исполнения Dedup, последовательной и параллельной версии, на файлах размера 159MB и 356MB**

На рисунке 8 представлены графики времени исполнения программы Dedup из набора приложений для тестирования производительности системы PARSEC и этой же программы в представлении Intel® Concurrent Collections. Dedup производит сжатие файлов методом deduplication. Алгоритм сжатия достаточно сложен, представляет собой конвейер, в котором элементы последней стадии, записи в файл, должны обрабатываться строго в последовательном порядке. Масштабируемость алгоритма в представлении Intel® Concurrent Collections невелика из-за большого числа необходимых зависимостей в графе.

Ведутся разработки по улучшению планировщика заданий путём назначения шагам приоритетов. Эта возможность позволяет уменьшить время исполнения программы Dedup на восьми в потоках в 1,4 раза.

## 8. Заключение

Была представлена новая модель параллельных вычислений, имеющая следующие цели:

- Поддержка всех типов параллелизма
- Поддержка всего спектра параллельных архитектур
- Поддержка всего спектра моделей исполнения
- Скрытие низкоуровневых вопросов параллельного программирования

Программа в этой модели состоит из абстрактного представления параллельного алгоритма и высокоуровневых операций и структур данных, реализованных на последовательном языке программирования.

Основными преимуществами использования описанного подхода к построению параллельных программ являются:

- Приложение легко портировать, потому что они не являются платформенно-зависимыми
- Более простая и быстрая разработка, что обеспечивается сокрытием вопросов параллелизма и выделением высокоуровневого описания алгоритма
- Вопросами производительности на конкретной платформе занимается реализация Intel® Concurrent Collections

Применение модели к стандартным тестам производительности показывает отличную масштабируемость в случаях, если вычисления не разбиваются на слишком мелкие шаги и зависимости по данным не образуют сложных графов с узкими местами в виде набора последовательных вычислений. Для более сложных случаев исследуется возможность автоматического объединения нескольких шагов и создания более эффективного планировщика вычислений.

## 9. Литература

[1] R. D. Blumofe and C. F. Joerg, “Cilk: An Efficient Multithreaded Runtime System”, Proceedings of the 5<sup>th</sup> ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 207-216. <http://supertech.csail.mit.edu/papers/cilkjpd96.pdf>.

[2] “OpenMP Application Program Interface, Version 2.5 May 2005”, from the OpenMP web site: <http://www.openmp.org>.

[3] J. Reinders, “Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism”, 2007.

[4] C. Bienia, S. Kumar, J. P. Singh and K. Li, “The PARSEC Benchmark Suite: Characterization and Architectural Implications”, Princeton University, TR-811-08.



# **The Integrated Enterprise Data Warehouse Engineering**

**Dr. Sergey V. Zykov, Ph.D.**

**TEKAMA LLC**

**4, 2<sup>nd</sup> Roschinskaya St., Moscow 115191 Russia**

**E-mail: [sergey.zykov@tekama.com](mailto:sergey.zykov@tekama.com)**

## **Abstract**

The paper considers content management in web-portals, embracing heterogeneous enterprise information systems (IS). Quite a heavy data burden has been accumulated by the enterprises, and it tends to increase further. Global distribution of the heterogeneous data and its weak-structuredness make it difficult to manage in a uniform way. A problem-oriented approach for enterprise content management is presented, which includes a formal model and a software development toolkit. Implementation results are given, which prove significant terms and cost reduction.

# **Интегрированное управление корпоративными хранилищами данных**

**Сергей Зыков, к.т.н., доц. ГУ-ВШЭ  
ООО «ТЕКАМА»**

**115191 РФ, Москва, 2<sup>я</sup> Рощинская ул, 4**

**E-mail: sergey.zykov@tekama.com**

## **Тезисы**

В работе рассматривается управление контентом на основе Интернет-порталов, объединяющих разнородные источники информации. Уже сегодня объемы корпоративных хранилищ данных весьма значительны, и они будут быстро возрастать в дальнейшем. Гетерогенность, глобальная распределенность и слабая структурированность этих данных затрудняют единообразное управление ими. Предложенный предметно-ориентированный подход к управлению корпоративными хранилищами данных включает как математическую модель, так и инструментальные средства поддержки разработки приложений. Полученные результаты подтверждают существенное сокращение сроков и стоимости внедрения систем управления корпоративными хранилищами данных.

## **1. Introduction**

State-of-the-art enterprise IS currently handle terabytes of heterogeneous information data (and metadata) sources, which tend to change for petabytes shortly. Such huge data volumes demand new models, methods and tools to manage them uniformly. A positive way of the solution is portal-based enterprise content management (ECM). Major software producers lack adequate formal models in their ECM schemes, which makes the process vendor-dependent and non-uniform. Thus, the approaches known as yet either have model-level methodological “gaps” or do not result in enterprise-level solutions with practically applicable implementation features (scalability, availability, fault tolerance etc.). The proposed ECM methodology is a part of the integrated IS lifecycle support for heterogeneous portal-based environment [12].

## **2. Theoretical background**

ECM methods used for combine statements of finite sequences [1], categories [2,3], semantic networks [4] and variable domains [5]. Finite sequences and categories are used for data object modeling. Categories are used to model virtual ECM machine by means of an abstract machine. Computations theory allows to construct denotation semantics of ECM. Semantic networks are used for visualization.

## **3. The methodology**

### **3.1 Outline**

The dynamic state-based ECM data model is adequate for heterogeneous weak-structured environments.

The model supports personalization-based front-end and back-end (meta)data processing. Component and event-based scripting technologies support extendable, distributed, and interoperable environments.

The variable domain-based model features event-driven (meta)data object management of heterogeneous problem domains. Therewith, the range of possible (meta)data sources is extended up to virtually arbitrary data warehouses that support both front-end globally distributed enterprise IS and legacy systems. The methodology features content-oriented (meta)data model based on an abstract machine.

A multi-level integrated IS design and implementation scheme is suggested (see fig. 1) that provides fast component-based ECM. The scheme is used to maintain the enterprise content adequacy and provide its integrity control. During the ECM lifecycle, the content specification is transformed from problem domain notations to formal computational data model entities, further, by means of a CMS-equipped CASE-toolkit to object-relational (meta)database scheme and, finally, to the target Web-representation. A formal language notation is suggested to represent the content and the environment during its lifecycle transformations.

### **3.2 The object model for data and metadata**

An object-based data model is suggested.

Data in the model is represented as follows:

Data = <class, object, value>,

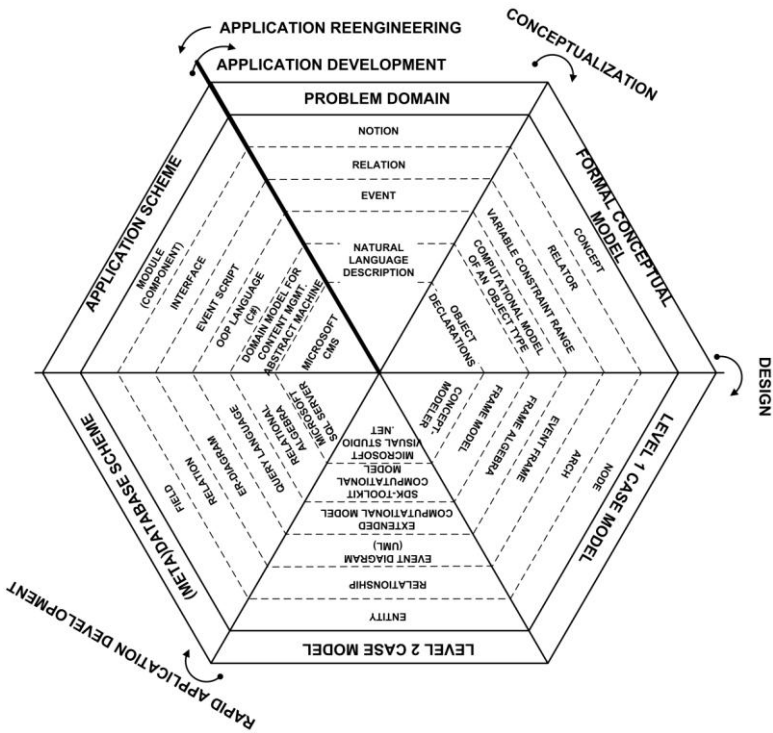


Figure 1. Generalized design and implementation procedure of IS for global network environments

where under a class a collection of objects of the integrated enterprise database is implied. An object means an element of ECM IS portal page template. A value means a template data object instantiation and represents the problem domain dynamics.

Compared to results known as yet, principal benefits of the model suggested are more adequate mapping of heterogeneous weak-structured problem domain dynamics and statics as well as event-driven (meta)data control in a global computational environment.

The data model is based on two-level conceptualization [6], i.e., the process of establishing relations between problem domain concepts.

Objects, according to assigned types, are assembled into assignment-dependent collections, thus forming variable domains, which model problem domain dynamics and statics.

(Meta)data and state semantics is adequately formalized by the model based on multi-sort typed lambda calculus, combinatory logic, semantic network scenarios and categorical abstract machine with states.

The data model compression principle allows model application to concepts, individuals and states separately and to the data on the whole.

The integrated object model for data, metadata and states is characterized by structural hierarchical organization, scalability, metadata encapsulation and readability. Extendibility, adequacy, neutrality and semantic soundness of the formalization provide problem-oriented IS development with (meta)data object adequacy maintenance throughout the entire lifecycle.

On the basis of multi-parameter functional

$$F = F((v), (e), \dots) (s) (p),$$

where assignment values represent:

- s – IS user personal preferences;
- p – IS user registration status;
- v – IS client interface parameters;
- e – IS data access device parameters,

a problem-oriented object model for portal personalization has been built, which is based on functional  $||F||$  evaluation function [8].

### 3.3 The model for content management

Abstract machine for content management (AMCM) [10] is suggested as an ECM IS model, which is an improved version of categorical abstract machine (CAM) [2]. At any given moment AMCM is determined by its state. AMCM work cycle can be formalized by explicit enumeration of possible state changes, which define the procedure of AMCM state dynamics modeling.

From the formal model viewpoint, when portal page templates are mapped into the pages, variable binding evaluates the variables that represent template elements and their values, i.e. portal page elements.

AMCM semantics can be described on the basis of D.Scott's variable domain theory [5]. Therewith, atomic template types are selected from standard domains, while more complex template types are built using domain constructors.

AMCM formal semantics is built as follows:

- 1) Standard (most commonly used within the model framework) domain enumeration;
- 2) Finite (containing explicitly enumerable elements) domain definition;

3) Domain constructor (operations of building new domains out of the existing ones) definition;

4) Aggregate domain formalization using standard domains and domain constructors.

Domain constructors include functional space  $[D_1 \rightarrow D_2]$ , Cartesian product  $[D_1 \times D_2 \times \dots \times D_n]$ , disjunctive sum  $[D_1 + D_2 + \dots + D_n]$  and sequence  $D^*$ .

Let the AMCM language contain expression set  $E$  (including constant set, identifier set  $I$ , assignment operation (content “write operation” to template “slot”) etc.), and command set  $C$  (comparison, command sequence etc.).

AMCM syntax is completely defined by the following syntax domain description:

$\text{Ide} = \{I \mid I - \text{identifier}\};$

$\text{Com} = \{C \mid C - \text{command}\};$

$\text{Exp} = \{E \mid E - \text{expression}\}.$

Let us collect all possible language identifiers into  $\text{Ide}$  domain, commands – into  $\text{Com}$  domain, and expressions – into  $\text{Exp}$  domain.

The state-based AMCM environment model can be represented as follows:

$\text{St} = \text{Mem} \times \text{In} \times \text{Out};$  (s)

$\text{Mem} = \text{Ide} \rightarrow [\text{Val} + \{\text{unbound}\}];$  (m)

$\text{In} = \text{Val}^*;$  (i)

$\text{Out} = \text{Val}^*;$  (o)

$\text{Value} = T_1 + T_2 + \dots$  (v)

AMCM state is defined by “memory” state considering input values (i.e. content) and output values (i.e. web-pages) of the abstract machine. Therewith, under memory a mapping from identifier domain into value domain is implied, which is similar to lambda calculus variable binding. For correct exception handling, unbound element should be added to the domains. Value domain is formed by disjunctive sum of domains, which contain content types of AMCM language.

Semantic statements describe denotates (i.e. correct construct values) of AMCM (meta)data object manipulation language.

Semantic statements for basic AMCM language commands and expressions are presented in [12].

Constant denotates are their respective values in a form of ordered pair of  $\langle \text{variable}, \text{value} \rangle$ , while program state remains unchanged.

Identifier denotates are identifiers bound with their values (if binding is possible) in a form of ordered tuples of <variable\_in\_memory, identifier, state>, while the state remains unchanged (an error message is generated if the binding is impossible).

Thus, ECM IS template binding with the content may result in AMCM state change and in a number of limited, predefined cases (particularly, under template and content type incompatibility) – in error generation.

Semantic statement for an AMCM command, which assigns content to template element, results in state change with substitution of content value by the identifier in memory.

#### **4. Customizing the methodology**

Let us apply the computational models introduced to the target ECM IS and the portal.

The problem domain model is based on two-level compression [6], which is interpreted here as establishing relationships between data object classes *C* of the integrated problem domain *D*, which, in general, are modeled by the domains:

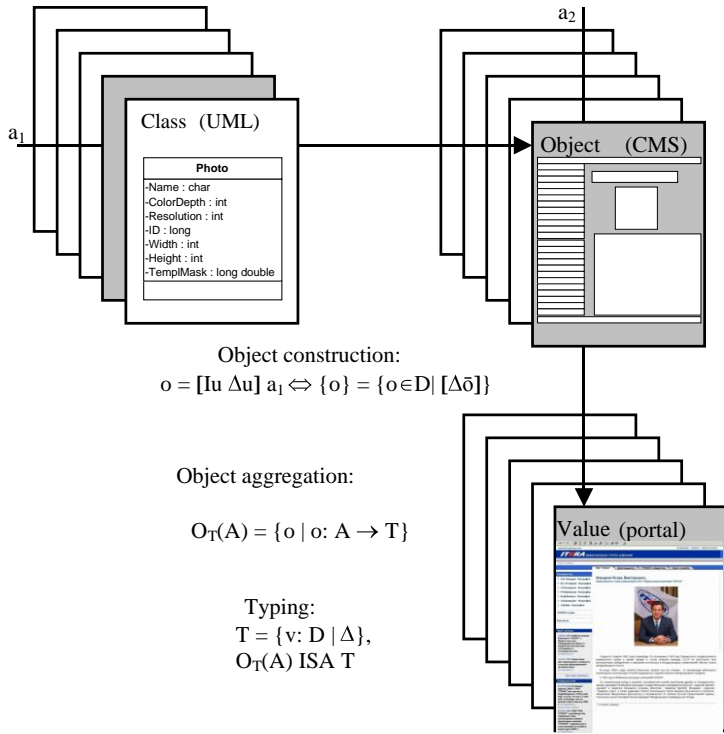


Figure 2. The CMS object model with a portal page generation example



$C = \text{Iw}:[D] \forall v:D (w(v) \leftrightarrow \Delta) = \{v:D \mid \Delta\},$   
 where:

C and D are in a relation of partial order with each other (C ISA D);

$\Delta$  is a condition of data object w belonging to class C (according to a problem domain expert).

Complex, “multi-dimensional” data objects are modeled as n-arity data object relationship (frame representation is given in [12]):

$$R^n = \text{Iw}: [V_1, \dots, V_n] \forall v_1:V_1 \dots \forall v_n:V_n (w[v_1, \dots, v_n] \leftrightarrow \Gamma) = \{[v_1:V_1, \dots, v_n:V_n] \mid \Gamma\}.$$

Thus, any class of data objects is a collection of ordered pairs  $(v_i:V_i)$ , where  $v_i$  is i-th attribute of the class, and  $V_i$  is type of the class.

However, class attributes contain not only data, but also metadata (e.g., object dimensions, and integrity constraints). Another metadata example is a number (binary bit mask), which defines effective rights of class objects usage in the templates of ECM IS.

Under assignment  $a_1$ , class C is instantiated with  $\Delta_k$  template of ECM IS web-page. Therewith, evaluation of template collection M assigns the value “true” to its only element  $m_i$ , the index of which equals to the template number (k):

$$M = (m_1, \dots, m_k, \dots, m_N),$$

where  $\forall i=1, \dots, N \ m_i \in \{0, 1\}$ ;

$$[M|\Delta_k] = (m_1^*, \dots, m_k^*, \dots, m_N^*),$$

where  $m_i^* = 1, i = k$  и  $m_i^* = 0, i \neq k$ .

Besides, the metadata attributes  $v_1, \dots, v_n$  are instantiated with metadata objects, according to constraint conditions  $t_i$  of template  $\Gamma$ :

$$[(v_1:V_1, \dots, v_n:V_n)]t_i = ([v_1]|\Gamma(t_1), \dots, [v_n]|\Gamma(t_n)) = (v_1':V_1', \dots, v_n':V_n').$$

Therewith,

$$V_1' \text{ ISA } V_1, \dots, V_n' \text{ ISA } V_n.$$

The second assignment  $a_2$  results in  $(v_1', \dots, v_n')$  template instantiation of the ECM IS web-page with content values of  $(c_1, \dots, c_n)$ :

$$[(v_1':V_1', \dots, v_n':V_n')]c = (v_1'/c_1, \dots, v_n'/c_n),$$

where

$$c_1:C_1, \dots, c_n:C_n,$$

and  $C_1 \text{ ISA } V_1', \dots, C_n \text{ ISA } V_n'$ .

Note that the abstraction level of the model elements decreases during transitions from classes to objects, and, further, to their values. Data object expandability during meta-level transitions provides extendable software development. According to the types assigned, the objects are aggregated into assignment-dependent collections and they form variable domains. Adequacy, neutrality and semantic soundness of

data objects and their components provide problem-oriented ECM with continuous model-level lifecycle support.

Object classes  $u$  are defined by means of description  $Iu \Delta(u)$  with the values of  $[Iu \Delta(u)]$ , where  $\Delta$  is a selection criterion. Applying assignments  $a_1 \in A$  and  $a_2 \in A$  from domain  $A$  transforms the classes first into objects

$$o = [Iu \Delta(u)] a_1,$$

and then to their values:  $c = o a_2$ .

Two-way assignment character (from classes to values and backwards) provides an adequate model of re-engineering; description mechanism simplifies modeling in both directions.

Variable domains

$$O_T(A) = \{o \mid o : A \rightarrow T\}$$

are constructed as collections of objects  $o$  with types  $T$ , extracted from problem domain  $D$  by selection criteria predicates  $\Delta$ , while the collection of possible data objects  $o$  is contained in  $D$ , and the collection of actual ones,  $O_T(A)$  is contained in  $T$ .

Thus, the basic modeling principle can be summarized as follows:

[ class of objects ] : assignment  $\rightarrow$  object,

where the bracketed part refers to the language level of class description, and the rest refers to the software level. The principle means that to declare a class we use selection criteria, which identify functions from assignments into objects, i.e. class is treated like a process.

Data objects are identified as follows:

[ object class ] : assignment  $\rightarrow$  object  $\triangleright$  object  $\triangleright \triangleright$  assignment  $\rightarrow$  value  $\triangleright$  value,

where " $\triangleright$ " means abstraction level decrease.

Thus, the diagram (see fig.2) illustrates the compression principle

$$o = [Iu \Delta u] a_1 \Leftrightarrow \{o\} = \{o \in D \mid [\Delta(\delta)]\},$$

and the essence of problem domain modeling process, which transfers language-level classes to problem-domain ones by the evaluation function  $[\bullet]$ . An object class is modeled by its selection criteria  $\Delta$  and its description  $I$ . Evaluation function maps problem domain objects to data definition language ones.

Figure 1 also gives an example of ECM modeling.

The example starts from data class (digital photo image) representation in UML. The class has the following image attributes:

ID: char;  
Name: int;  
ColorDepth: int;  
Resolution: long;  
Width: int;  
Height: int;

TemplMask: long double;

The latter attribute refers to the bit mask of the ECM template.

The first assignment  $a_1$  fixes certain template attributes (such as relative position of the digital photo image, etc.) in the resulting enterprise portal web page. The second assignment  $a_2$  results in evaluation of the ECM template web page by the content values.

IS development methodology has been customized for enterprise portal management, including information and interface (i.e. data and metadata) of Internet and Intranet sites. Detailed design scheme is given in [11].

According to the scheme, a formal procedure for heterogeneous data warehouse processing is suggested that allows users to interact with the integrated distributed (meta)database in a certain state, depending upon dynamical script-activated assignments. Therewith, scripts depend on user-triggered events and provide transparent intellectual client-server front-end interaction. Dynamic profiles for (meta)database access provide strict and flexible personalization, high fault tolerance and data security for ordinary and privileged IS users in heterogeneous environment.

## **5. Implementation overview**

The ECM development approach has been practically approved by constructing Internet and Intranet portals for ITERA International Group of Companies (<http://www.iteragroup.com>).

The Menu module of ECM is aimed at portal navigation data storage and processing. Pages module is related to Menu and tracks events of assignment, migration and deletion of portal pages, (meta)data and navigation menu items. Images module provides storage, retrieval and portal web publication of digital photo and graphics images. News Columns module supports periodical portal publications (press releases, media news, etc.) including data from related third-party IS modules. Special sections module organizes visual management of portal content (i.e. data) and design (i.e. metadata) by given criteria set. Administration module implements data personalization, profiling, access control policies and (meta)database synchronization.

In terms of system architecture, the ECM IS provides assignments (depending on front-end position in data access hierarchy) with assignments for (meta)data entry, modification, analysis and report generation (from administrator level down to reader one). Problem-oriented form designer, report writer, online documentation and administration tools make an interactive interface toolkit. (Meta)database supports integrated storage online access to data and

metadata (e.g., object dimensions, integrity constraints, representation formats, etc.).

Implementation process included fast prototyping (with MySQL DBMS and Perl scripting) and full-scale integrated Oracle-based implementation. The fast IS prototype proved adequacy of the (meta)data model and of the approach on the whole.

Upon prototype testing, the ECM IS has been developed, which manages Intranet and Internet portals (<http://www.iteragroup.com>).

Implementation proved 40% terms and costs reduction (on the average) as compared to commercial software available. Functional features have been essentially improved and include better ERP and legacy IS integration, easier handling complicated objects and smarter report generation. Advanced personalization and access control have substantially reduced risks of (meta)data damage or loss [9].

## **6. Results, conclusion and future work**

The ECM construction methodology with lifecycle support has been introduced. The methodology is a part of the integrated approach to enterprise IS development [8], which provides adequate, consistent and integrate (meta)data manipulation during its entire lifecycle.

A set of (meta)data object models have been constructed. It includes state-based dynamical models for problem domain and development tools. The models provide integrated (meta)data object manipulation in weak-structured heterogeneous problem domains.

An ECM IS for (meta)data object management software development toolkit has been implemented. The IS has content-based architecture with front-end and back-end interfaces.


A fast event-driven prototype and the full-scale ECM IS have been implemented. The IS manages Internet and Intranet portals for a corporation employing around 10,000 people in nearly 150 companies of more than 20 countries. Implementation results proved substantial terms and costs reduction as compared to commercially available software. The ECM IS is based on a model, which integrates objects management methods for data and metadata.

The author is going to continue his studies of models and tools that support ECM systems.

## **References**

- [1] Barendregt H.P. The lambda calculus (revised edition), Studies in Logic, 103, North Holland, Amsterdam, 1984.
- [2] Cousineau G., Curien P.-L., Mauny M. The categorical abstract machine. In Science of Computer Programming 8(2): 173-202, 1987.

- [3] Curry H.B., Feys R. Combinatory logic, Vol.I, North Holland, Amsterdam, 1958.
- [4] Roussopoulos N.D. A semantic network model of data bases, Toronto Univ., 1976.
- [5] Scott D.S. Domains for denotational semantics. In ICALP 1982, 577-613.
- [6] Wolfengagen V.E. Event-driven objects. In Proceedings of the 1<sup>st</sup> International Workshop on Computer Science and information Technologies CSIT'1999. MEPhI Publishers, Moscow, Russia, 1999, Vol.1. p.p. 88-96.
- [7] Zykov S.V. Human Resources Information Systems Improvement: Involving Financial Systems and Other Sources Data. In: Advances in Databases and Information Systems, Springer, 1998, p.p. 351-356.
- [8] Zykov S.V. The Integrated Approach to ERP: Embracing the Web. In: Proc. CSIT'2002. Patras, Greece, 2002.
- [9] Enterprise Content Management: Bridging the Academia and Industry Gap In: Proc. of International Conference on Information Society (i-Society 2007), Merrillville, Indiana, U.S.A., October 7-11, 2007, Vol. I, p.p.145-152.
- [10] Zykov S.V. Enterprise Content Management: the Integrated Methodology. In: Enterprise Information Systems and Web Technologies (EISWT'07), Orlando, FL, U.S.A., July 9-12, 2007, p.p. 226-233.
- [11] Zykov S.V. An Integral Approach to Enterprise Content Management. In: N.Callaos, W.Lesso, C.D.Zinn, B.Zmazek (Eds.), Proc. of International World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI 2007), Orlando, FL, U.S.A., July 8-11, 2007, Vol. I, p.p. 212-216 216.
- [12] Zykov S.V. Integrated Methodology for Internet-based Enterprise Information Systems Development. In: WEBIST 2005, Proceedings of the First International Conference on Web Information Systems and Technologies, Miami, Florida, USA, May 26-28, 2005. Setubal, INSTICC Press, 2005, pp. 168-175.



Published by  
TEKAMA  
Moscow, Russia  
October, 2008

All copiright remain with the autors  
<http://secr.ru>