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 (English)

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

Abstract (Russian)

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). 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 проектов 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 оправданна.

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.

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 циклам 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

Практика CMMI	Разработка Требований , документирование	Управление Требованиями	Управление Конфигурацией	Технические Решения - 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
Невысокие требования к качеству	++	++	++	++	++	++
Мало сотрудников, работа в одном месте	+	+	+	++	+	+
Требования изменчивы	++	+++	+	++	++	++

важность и полезность применения практик для условий, определенных в левом столбце.

Как видно из таблицы, CMM/CMMI 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, авионика, 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 CMMI 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, старт-апов, 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) 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), еженощное построение продукта и выполнение unit tests.

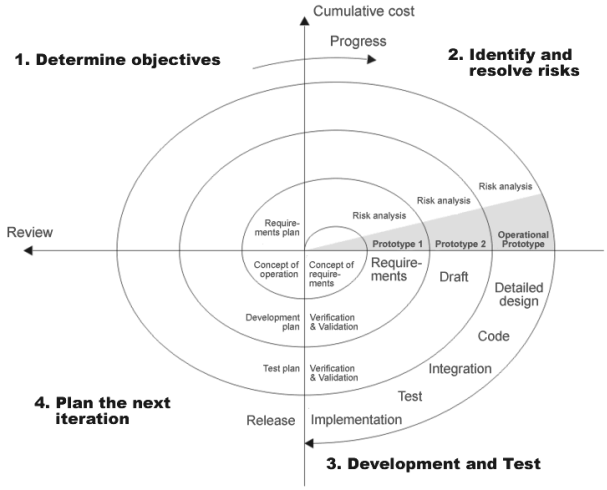
Однако, примерно через полгода стали нарастать проблемы… Первоначальной задачей было изготовление прототипа, своего рода proof of concept. Это было сделано, и на этом этапе взаимодействия квалифицированных инженеров между собой хватало для понимания текущих задач и их успешного выполнения. Но появлялись 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 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, нередки 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 состояния trunk.

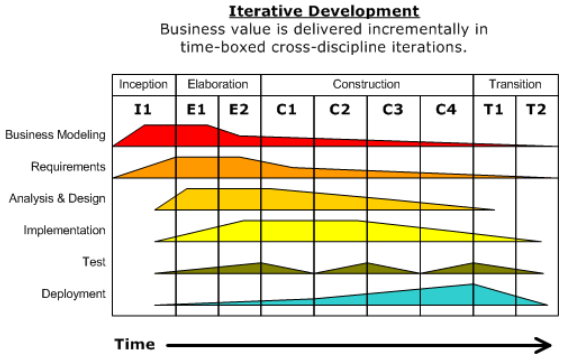
Всё это элементы из практик CMMI, они вводились и продолжают добавляться по мере необходимости. Их введение позволило значительно улучшить обмен информацией внутри отдела, избавиться от простоев в случае нестабильного trunk, улучшить планирование и управляемость проекта. На мой взгляд, это именно пример 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:



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.

CMMI 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 CMMI «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 CMMI. Так что, неприятие некоторыми горячими сторонниками Agile методологий CMMI может быть лишь свидетельством не очень хорошего понимания этой области.

5.1. Добавляем CMMI 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…

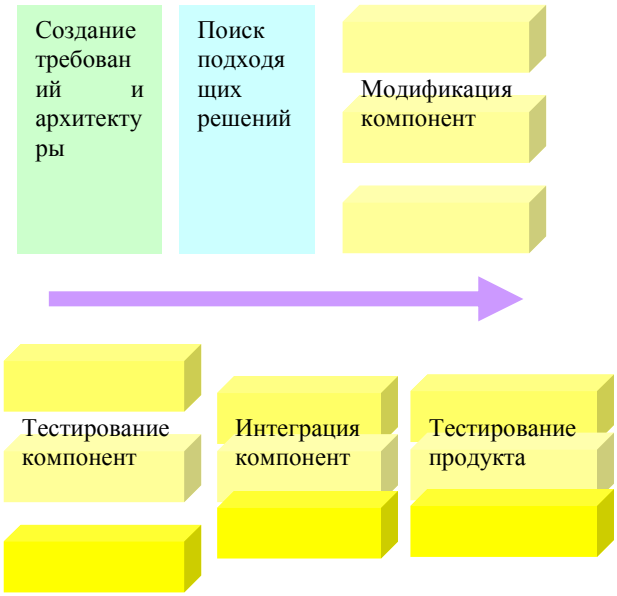
- Написание дизайна затянулось, и сорваны сроки начала кодирования? (Помогло бы планирование начала кодирования сразу после принятия основных решений об архитектуре, вместе с уточнением деталей дизайна, в несколько итераций)
- Группа тестирования говорит что не может начать тестовый цикл, так как разработчики 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[1]: «…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 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.

[1] 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