Эволюционный дизайн

Денис Миллер
dmiller@agileconsulting.ru

Abstract (Russian)

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 становиться 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 Беком 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