Agile, SCRUM и итерационная разработка по RUP

Кумсков М.И.

Профессор Департамента Программной Инженерии Факультета
Компьтерных Наук НИУ ВШЭ
Профессор кафедры Вычислительной Математики Мехмата МГУ им.М.И.Ломоносова
Эксперт Учебного Центра Люксофт
kumskov@mail.ru

Аннотация. Рассмотрены известные недостатки итерационной разработки и их преодоление в Agile разработке. Показано, что половина преимуществ, декларируемых в Agile манифесте достигается за счет итерационной разработки. В agile проектах используется менеджмент, называемый процессным, позволяющий самоорганизоваться проектной команде. Утверждается, что предварительное использование итерационной разработки с соответствующей инструментальной поддержкой работ в проекте является залогом последующего успешного перехода на agile разработку в проектах Time and Material (PMBoK).

Abstract. Agile Manual summarizes common iterative development disadvantages and their troubleshooting. Half of the advantages, declared in the Agile Manifest, are proved to be reached due to iterative development. Agile Projects make use of so-called processing management, enabling the Project Team to self-organize. It is claimed, that iterative development initial use with the corresponding project work tool support is a key to successful switch to Agile development in Time and Material (PMBoK) Projects.

Проекты разработки программного обеспечения (ПО) – статистика за последние 20 лет удивительно стабильно указывает, что успешные проекты составляют всего не более трети от всех проектов [1]. Можно назвать ключевую причину неуспеха ИТ-проектов, вспомнив как метафору известный из Библии «проект» построения Вавилонской башни. Ограничений на ресурсы не было, ограничений на время не было – но из-за смешения языков строителей – т.е. из-за нарушения «проектных коммуникаций», – проект был неуспешен. Итак, нарушение проектных коммуникаций – это основная причина неуспеха ИТ-проекта.

Нарушение **внешних** проектных коммуникаций – это риски неверного понимания потребностей заказчика, риски неверной концепции проекта, риски неверного определения требований с системе. Нарушение **внутренних** проектных коммуникаций – это риски неверных коммуникаций в проектной команде, это риски неверной внутренней декомпозиции системы на составные части, это архитектурные риски, связанные с отсутствием «чертежей» разрабатываемой системы (нет визуального моделирования). Если говорить кратко, то есть 2 основных источника рисков в ИТ-проекте – это риски концепции и требований, а также архитектурные риски.

Проблемы нарушения проектных коммуникаций обостряются при масштабировании – в небольших проектах (3-5 человек) все просто и понятно, коммуникации относительно «прозрачны», а коммуникационные инциденты устраняются относительно незатратно. Для поддержки адекватных проектных коммуникаций **при масштабировании проекта** были преложены «лучшие практики» (best practice) как паттерны организации работ в проекте [2]. Этих практик шесть и в своей первоначальной формулировки они выглядели так – это:

1. Итерационная разработка
2. Управление требованиями
3. Использование компонентной архитектуры
4. Визуальное моделирование
5. Постоянная проверка качества (включая тестирования)
6. Управление изменениями и конфигурациями

Приведем краткую характеристику лучших практик, чтобы сравнить их с принципами agile проектов.

Итерационная разработка (есть в Agile). Итерация называется спринтом и длиться две недели). Это выполнение проекта как серии мини проектов, называемых итерациями. Все функциональные требования приоритезируются и разбиваются на группы, в первую итерацию итерацию выбираются архитектурно значимые функциональные требования. Итерация выполняется «по каскаду» и всегда заканчивается работающим прототипом системы, готовым к демонстрации заинтересованным лицам. Реализуется только часть функционала системы(т.е. первая группа требований). Итерация имеет план, который как правило не меняется в ее ходе и критерий успеха итерации. Это самая мощная и самая трудная для внедрения практика.

Управление требованиями (есть в Agile) – это наличие в проекте двух регламентов (формальных или неформальных) работы с требованиями. Первый регламент описывает порядок выявления требований (в Agile – это «user story»), структурирования и документирования требований. Второй регламент определяет как проводятся работы с изменяющимися требованиями (есть в Agile).

Использование компонентной архитектуры – (нет в Agile). Компонента – это модуль, публикующий свои интерфейсы. Компонентная архитектура предполагает, что модули «общаются» друг с другом только через интерфейсы. Если модуль обновляется (например, в результате исправления ошибки), то интерфейс «переопубликовывается». Другие компоненты продолжают пользоваться известным им интерфейсом, но для исполнения теперь будет вызываться уже обновленный модуль. Таким образом на системном уровне поддерживается принцип «полиморфизма» – разные реализации одного интерфейса. (*Метафора*: электронные приборы имеют разъемы на плате – это интерфейсы, вставляемые чипы – это модули).

Визуальное моделирование (UML Unified Modeling Language) – (нет в Agile). Использование графической нотации, чтобы (1)показать состав модулей программной системы, отношения между ними и «в статике» и «в динамике», когда модули обмениваются сообщениями друг с другом; (2)на основе исполь-

зования CASE средств по модели на UML автоматически генерировать «каркасный код» приложения (преобразование называется «прямое проектирование» – forward engineering) и, обратно, восстанавливать визуальную модель по имеющемуся объектно-ориентированному коду приложения (преобразование называется «обратное проектирование» – reverse engineering); (3)сохранять и повторно использовать паттерны проектирования – стандартные и «хорошие» решения стандартных часто встречающихся задач []; (4) документировать проектные решения, принимаемые разработчиками и архитектором в зоне их ответственности при работе с компонентами и структурой базы данных. (*Метафора*: если вы строите многоэтажное здание (сложную систему), то нужны его чертежи, включая поэтажные планы).

Постоянная проверка качества (включая тестирования) (есть в Agile). Осуществляется проверка качества выполненных работ (quality assurance), включая ревью (требований, архитектуры, стиля программного кода). В Agile – это еще и проведения этапа «ретроспектива спринта». Осуществляется проверка качества продукта – системное тестирование (quality inspection).

При итерационной разработке возникает технологическая проблема при проведении тестирования – появляется высокий объем регрессионного (повторного) тестирования функционала, разработанного на предыдущих итерациях. Для преодоления проблемы тестирование должно быть автоматизировано. Эта важная особенность проверки качества в agile не упоминается – предполагается что agile команда достаточно профессиональна.

Управление изменениями и конфигурациями. Предполагается, что все запросы на изменения (CR – Change Request, RfC – Request for Change) накапливаются в едином репозитории, приоритизируются и оцениваются командой (есть в Agile). Трудоемкие запросы рассматриваются совместно с заказчиком в рамках специальных встреч комитета по изменениям (CAB – Change Advisory Board), чтобы пересмотреть приоритеты ранее определенных требований и, возможно, сократить объем ранее согласованных работ (Scope). Для работы с изменениями и дефектами используется специальный инструмент. Управление конфигурациями (нет в Agile) связано с пользованием в проекте версионного контроля проектных артефактов, включая программный код модулей. При проведении сборки выбирается одна из версий каждой компоненты и *конфигурацией* называется «набор версий компонент вошедших в сборку». Конфигурация описывается как BoM-файл (Bill of Material) и его наличие позволяет точно повторить именно ту сборку, которая была ранее. Для работы с конфигурациями используется специальный инструмент.

Когда молодые руководителя проектов узнают о лучших практиках, они обычно просят более подробно остановиться на особенностях их внедрение в проектную работу. Паттерны по реализации этих практик вместе с их инструментальной поддержкой описаны в информационном источнике, который теперь называется IBM Rational Unified Process (RUP) [3]. Методика внедрения лучших практик на основе RUP «хорошо известна в узких кругах» методологистов и описана в книге Кролла и Кратчена, переведенной на русский язык [4].

Гибкие методологии. Важный аспект методологического подхода в проекта разработки ПО – это *тип управления*, принимаемый в проекте – задачный или процессный. Классический (или задачный) менеджмент отличается от процессного менеджмента тем, как определяется критерий успеха выполнения работ.

Задачное управление. Если сотрудник вовремя и качественно выполнил порученную ему задачу, то он «хорошо» работает и заслуживает поощрения. По умолчанию предполагается, что если все сотрудники будут так хорошо работать в проекте, то проект будет успешным. При этом для сотрудника не важно, как дальше будут использованы результаты его задачи – за это отвечает начальство (руководитель проекта – РП).

Процессное управление. Здесь оценка работы сотрудника проводится совсем иначе. Можно воспользоваться как метафорой девизом мушкетеров: «один за всех и все за одного». Это значит, что если все цепочка работ процесса выполнена в срок и качественно, то все работали «хорошо». Но если в цепочки работ произошел сбой – кто-то не так выполнил работу и процесс «не уложился» в требуемые метрики «хорошести» (в срок и качественно), то виноваты все. Процесс обслуживает «клиента» – инициатора процесса, – которому не важно, почему он был некачественно обслужен. Из-за этого процессное управление иногда называют «ориентированное на клиента» или «ориентированное на качество», а метрики процесса – цепочки его работ, – называют KPI – Key Process Indicator. Сокращение KPI иногда переводят как Key Performance Indicator, тем самым делается попытка приспособить терминологию процессного управления к задачному менеджменту и KPI назначают на исполнителей задач, что сводит на нет преимущества процессного управления.

Раннее уже отмечалось, что внутри Agile проектов используется процессное управление, важным принципом которого является ориентация всех участников на постоянное улучшение процесса (Continues Process Improvement – CPI). В Agile эти улучшения предлагаются и принимаются к реализации на фазе спринта «ретроспектива».

Первоначально процессное управление было предложено и на практике реализовано Эдвардом Демингом в японской автомобильной компании Toyota. Он ввел понятие цикла управления PDCA (Plan – Do – Check(KPI) – Act(улучшение процесса)) и ввел понятие «владельца процесса», который отвечает за его улучшение и соответствие метрикам KPI.

В agile командах внутренние процессы определяются (и совершенствуются) на основе «самоорганизации» (девиз мушкетеров). Поэтому руководство и заказчик «должны» договариваться с командой на основе партнерских отношений – здесь они «не могут командовать» в смысле задачного управления. Такие отношения весьма непривычны для классических руководителей проектов, обученных и привыкших к отношениям «начальник – подчиненный» в парадигме задачного менеджмента согласно PMBoK (Project Management Body of Knowledge). Руководители привыкли выдавать задачи, контролировать ход их выполнения и сроки. Теперь же нужно доверять исполнителям, договариваться с ними, что требует перестройки проектных коммуникаций – как с психологической, так и с практической точки зрения. Не всегда руководители к этому готовы.

Таблица 1. Основополагающие принципы Agile-манифеста и их связь с итерационной разработкой

Основополагающие принципы Agile-манифеста	Итерационная разработка	Самоорганизация	Лучшие практики
Наивысшим приоритетом для нас является удовлетворение потребностей заказчика, благодаря регулярной и ранней поставке ценного программного обеспечения.	X		
Изменение требований приветствуется, даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества.	X		
Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев.	X		
На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.	X	X	
Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.		X	
Непосредственное общение является наиболее практичным и эффективным способом обмена информацией как с самой командой, так и внутри команды.		X	
Работающий продукт – основной показатель прогресса.	X		
Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм бесконечно. Agile помогает наладить такой устойчивый процесс разработки.	X	X	
Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.		X	X
Простота – искусство минимизации лишней работы – крайне необходима.		X	
Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.		X	
Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.		X	

В таблице 1 представлена связь «Основополагающих принципов Agile-манифеста» [5] с итерационной разработкой. Из таблицы видно, что половина принципов Agile обеспечивается за счет ***итерационной разработки*** как мощного средства улучшения внешних проектных коммуникаций.

Про недостатки классических итераций

К недостаткам итерационной разработки можно отнести следующие три аспекта.

Первое. Классическая итерационная разработка требует чтобы составлялись новые планы для каждой итерации, в которые включаются задачи по реализации функциональных требований и возникших запросов на изменения. Можно сказать, что нагрузка на руководителя проекта значительно возрастает – в конце итерации от него требуется создание плана на следующую итерацию. Однако известно, что «начальство» (руководителя проекта в нашем случае) не любит больше работать, поэтому руководитель проекта не заинтересован выполнять проект итерационно – каскадная разработка значительно «комфортнее», поскольку нет интенсивного планирования каждую итерацию. В Agile эта «проблема» успешна решена – команда сама планирует свою работу в спринте и ей доверяют относительно оценок трудоемкости задач в спринте (итерации).

Второе. При переходе от каскадной разработке к итерационой заказчик не всегда готов к такому переходу. Фактически заказчику нужно в конце каждой итерации «выделять ресурс», т. е. посылать своего представителя, имеющего полномочия принимать решения по продукту и его свойствам, для оценки результатов итерации. Этот представитель заказчика в ходе демонстрации текущего прототипа программой системы «выставляет» запросы на изменения и устанавливает их приоритеты. В рамках проектов Fix-Price заказчик несет финансовые потери, периодически отдавая в проект «совсем не дешевого» сотрудника для оценки промежуточных прототипов и его **свойств**. Это не всем нравиться – известна следующее высказывание заказчика: «У нас нет времени смотреть ваши хакерские недоделки. Мы будем с вами разговаривать в сроки, когда по плану будут приемо-сдаточные испытания. До свидания». В Agile эта проблема преодолена, поскольку проект выполняется как Time-and-Material и все риски лежат на стороне заказчика.

Третье. Системное тестирование в конце итерации предполагает проведение и регрессионного тестирования, т.е. повторного тестирования функционала, ранее уже проверенного в предыдущих итерациях. Это означает, что если в Agile проекте системное тестирование не автоматизировано, то нагрузка на тестировщиков нарастает при регрессионном тестировании от спринта к спринту. Справиться с таким объемом работ тестирования, мягко говоря, не просто – практический выход в ом, чтобы «сократить объем регрессионное тестирования» и снизить качество продукта.

Недостатки классического Agile

Первое. В классическом Agile команда не ориентирована на минимизацию архитектурных рисков, если в проекте разрабатывается *новый продукт.* В этом случае на первых спринтах должна сложиться и быть проверена архитектура системы, которая должна быть затем утверждена (baseline). Такая ситуация неявно предполагает, что Agile команда достаточно зрелая и профессиональная, чтобы справиться с архитектурными неопределенностями первых спринтов. Как следствие, вопросы визуального моделирования не входят в рекомендации Agile, – считается команда проекта понимает риски нарушения внутренних коммуникаций при отсутствии «чертежей» внутреннего устройства продукта.

Второе. В Agile проекте должны использоваться средства автоматизации тестирования и быть обученные и опытные тестировщики – об этом говорилось выше. Явно об этом в Agile принципах и практиках не сказано – предполагается, что команды профессиональна и все это у нее есть.

Третье. Желание использовать Agile универсально везде и всюду, во всех типах проектах разработки программного обеспечения, включая Fix-Price проекты, где все риски на стороне команды разработки.

Выводы

Из предыдущего обсуждения можно сделать интересные выводы. Agile разработку следует использовать в зрелых профессиональных командах, имеющих навыки оценки и планирования задач, владеющих средствами автоматизацией тестирования, использующих инструменты конфигурационного управления и средства управления задачами и дефектами, инструменты проектирования архитектурных решения на основе компонент и средств визуального моделирования.

Чтобы вырастить у себя в организации такие команды (в прошлом такие сплоченные самоорганизованные команды называли RAD группами [6] (RAD – Rapid Application Development)) следует вначале освоить классическую итерационную разработку и внедрить у себя лучшие практики, включая инструментальную поддержку работ по этим практикам.

Литература

1. Zucker Alan What We Really Know About Successful Projects – https://www.scrumalliance.org/ community/articles/2016/october/what-we-really-know-about-successful-projects
2. Best Practices for Software Development Teams – https://www.ibm.com/ developerworks/rational/ library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf
3. Крачтен Ф. «Введение в Rational Unified Process». – Пер. с англ., 2-е изд. М.: Вильямс, 2002.

4. Кролл П., Крачтен Ф. «Rational Unified Process – это легко. Руководство по RUP для практиков». – Пер. с англ. – М.: КУДИЦ-ОБРАЗ, 2004.
5. «Основополагающие принципы Agile-манифеста» – http://agilemanifesto.org/iso/ru/principles.html
6. RAD (программирование) – https://ru.wikipedia.org/wiki/RAD_(программирование)