Agile-манифест[1] довольно прост. Он состоит из 9 строк и включает 12 принципов, но значение его огромно.

До его принятия бытовало общее мнение (за редкими исключениями), что если вы разрабатываете что-то «серьезное», вы должны использовать водопадные техники, ориентированные на конечный продукт.

Методология agile оказалась прорывной, и сейчас именно она, а не водопадная модель является доминирующей, по крайней мере в мышлении.

Однако многие компании до сих пор тяготеют к водопадной модели — если не в техническом плане, то в организационном.

Тем не менее agile-методология имеет более прочные основы, чем ее предшественники. Их можно сформулировать как «инспекция и адаптация» (Inspect and Adapt, I&A).

Подобное изменение взглядов было значительным, но недостаточным. Почему значительным? Потому что оно ознаменовало стремление трактовать разработку как задачу познания, а не производства. Водопадная модель может быть эффективной для некоторых задач, но она проигрывает там, где требуется исследование.

Хотя повышение производительности на порядок, о чем говорил Фред Брукс, и кажется невозможным с точки зрения технологий, инструментов и процессов, некоторые подходы настолько неэффективны, что усовершенствовать их в разы очень даже реально. Водопадная разработка — именно такой случай.

Водопадный способ мышления основан на предположении, что «если мы хорошо подумаем/поработаем, то сразу же сделаем то, что нужно».

Agile переворачивает подобные представления. Он утверждает, что мы обязательно сделаем все не так. «Мы не понимаем, чего хочет пользователь», «мы не создадим нужный дизайн с первой попытки», «мы не знаем, исправили ли мы все баги в коде», и так далее и т. п. Команды, работающие по принципам agile, исходят из того, что они совершают ошибки, и тем самым снижают стоимость этих ошибок.

Agile разделяет научные принципы. Именно научному подходу свойственно критическое рассмотрение идей и стремление доказать их ошибочность, а не верность (опровергнуть).

Эти два способа организации мышления, основанные на предсказуемости и исследовании, формируют абсолютно разные, несовместимые подходы к управлению проектами и работой команд.

[1] Agile-манифест, https://agilemanifesto.org/.

Согласно принципам agile, мы организуем работу команд, процессы и технологии так, чтобы ошибки не имели серьезных последствий, их было легко выявлять, исправлять и в идеале избегать в дальнейшем.

Споры о преимуществах Scrum перед экстремальным программированием, или непрерывной интеграции перед ветвлением функционала, или разработки через тестирование перед тщательным обдумыванием кода не имеют смысла. Любой действительно гибкий процесс – это эмпирическое управление процессом.

Такой подход к разработке гораздо эффективнее, чем предшествующий ему, основанный на конечном продукте прогнозный метод водопадной разработки.

Итеративный подход принципиально отличается от последовательной работы по плану. Он значительно более эффективен.

Многим читателям это покажется очевидным, но это не так. История разработки показывает, что раньше итерации считались необязательными, а основой успешной работы было тщательное предварительное планирование всех ее этапов.

Итерация лежит в основе всей исследовательской деятельности и является необходимым условием приобретения действительных знаний.

ПРАКТИЧЕСКИЕ ПРЕИМУЩЕСТВА ИТЕРАТИВНОГО ПОДХОДА

Если мы рассматриваем разработку как познание и открытие, то в ее центре должна находиться итерация. Однако преимущества итеративного подхода не всегда очевидны с первого взгляда.

Возможно, самое важное — то, что переход к итеративному способу организации работы автоматически сужает наше поле зрения и побуждает думать категорийно, уделяя внимание модульности и разделению ответственности. Эти понятия возникают как прямое следствие итеративного подхода, но в конце концов становятся частью логического механизма повышения качества нашей работы.

Одна из идей, общих для Scrum и экстремального программирования, — это необходимость деления рабочего процесса на небольшие этапы. Логика

agile такова: прогресс в разработке трудно измерить, но мы можем измерить завершенный функционал, поэтому следует работать с отдельными функциями, чтобы видеть результат.

Подобное ограничение стало значительным шагом вперед. Однако процесс усложняется, если вы хотите понять, сколько осталось до «завершения». Такой итеративный подход к разработке отличается от традиционных методов. Например, при непрерывной доставке мы готовим незначительные изменения к релизу по нескольку раз в день. Они должны быть настолько завершенными, чтобы релиз продукта оставался надежным и безопасным в любой момент. Так что же в действительности означает «завершенный» в этом контексте?

Каждое изменение завершено, поскольку оно готово к релизу, поэтому единственной осязаемой метрикой окончания изменения является ценность, которую это изменение создает для пользователя. Это очень субъективно. Как узнать, сколько коррективов понадобится, чтобы создать ценность для пользователя? Большинство компаний-разработчиков стремятся предоставить набор функций, которые в совокупности представляют «ценность», но если изменения возможны в любое время, то эта концепция становится довольно размытой.

Определить, какие именно изменения создают «ценность», сложно, поскольку это предполагает, что, приступая к работе, вы уже знаете, какие функции вам понадобятся, и способны определить, насколько вы продвинулись к цели, то есть к «завершенности». Это очень упрощенное изложение того, что имели в виду пионеры agile-методологии, но идея, которой руководствуются большинство традиционных компаний при переходе к agile-планированию, именно такова.

Одно из самых неочевидных преимуществ итеративного подхода в том, что у нас есть выбор. Мы можем совершать итерации, работая над продуктом, и управлять им на основе обратной связи от пользователей, чтобы повысить его ценность. Это один из наиболее значимых аспектов этого подхода, и его часто упускают из виду при внедрении такого способа работы.

Тем не менее, независимо от намерений или результатов, пакетно-ориентированный подход помог нашей отрасли уменьшить объемы и сложность функций, которые мы создаем, и это действительно важный шаг вперед.

Agile-планирование во многом основано на разбиении рабочего процесса на небольшие этапы, что позволяет завершить работу над одной функцией в рамках одного спринта, или итерации. Изначально это был способ измерения прогресса, но он оказался очень полезен с точки зрения получения точной и постоянной обратной связи о качестве и целесообразности нашей работы. В результате повышается скорость нашего обучения. Работает ли этот дизайн? Нравится ли эта функция пользователям? Достаточно ли быстро функционирует система? Все ли ошибки мы исправили? Грамотно ли написан код? И так далее.

Итерации небольших, четко определенных и готовящих продукт к релизу шагов помогают получить ценную обратную связь.

ИТЕРАЦИЯ КАК СТРАТЕГИЯ ЗАЩИТНОГО ПРОЕКТИРОВАНИЯ

Итерации способствуют внедрению защитного проектирования (defensive design). Мы рассмотрим этот подход подробнее в части III.

Интересную точку зрения на принципы agile впервые высказал мне мой друг Дэн Норт. Он представил различие между водопадным программированием и agile с точки зрения экономики. Водопадная разработка исходит из того, что с течением времени стоимость изменения растет. Это классическая интерпретация модели «Стоимость изменений» (рис. 4.1).

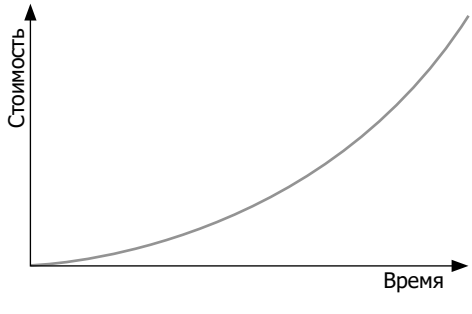

Рис. 4.1. Классическое представление стоимости изменений

Такое представление сопряжено со сложностями: если модель верна, то единственный разумный выход — принимать все важные решения

в начале работы над проектом. Но проблема в том, что на первоначальном этапе наши знания минимальны. Поэтому, как бы мы ни старались, мы принимаем ключевые решения, основываясь на ничем не подкрепленных предположениях.

Какой бы предварительный анализ мы ни провели, проект никогда не начнется с «...мы поняли, что нам делать, от и до». И учитывая, что мы никогда не будем иметь на старте «четко определенный набор исходных данных», вне зависимости от тщательности составления планов, регламентированная процессная модель, или водопадная разработка, даст сбой при первой же сложности. Невозможно вписать разработку ПО в столь неподходящий для нее шаблон.

Неожиданности, недопонимания и ошибки — обычное дело в разработке, потому что это часть процесса исследования и открытий, а значит, нам необходимо уделять больше внимания обучению, чтобы избегать ошибок, которые обязательно встретятся на нашем пути.

Вернемся к мысли Дэна Норта: если классическая модель стоимости изменений неэффективна, то что тогда? Что, если мы сделаем кривую этой модели более плоской (рис. 4.2)?

Предположим, мы изменим подход так, чтобы предлагать новые идеи, выявлять и исправлять ошибки на одном уровне стоимости независимо от того, на каком этапе это будет происходить. Кривая стоимости станет плоской.

Это даст нам свободу открытий и преимущества от использования открытий. Мы сможем непрерывно совершенствовать знания и код, а также опыт взаимодействия пользователя с нашим продуктом.

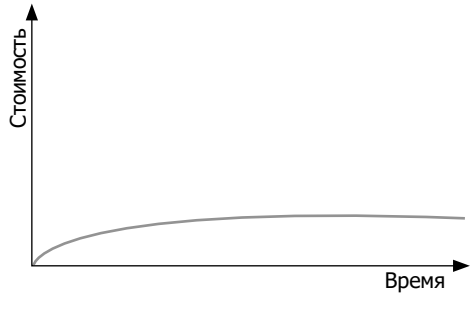

Рис. 4.2. Представление стоимости изменений в модели agile

Итак, что нужно, чтобы выровнять эту кривую?

Нам не удастся уделять больше времени исключительно анализу и проектированию, без реального процесса создания, поскольку это подразумевает меньше времени на изучение. Поэтому нам необходимо уплотниться, надо работать итеративно. Анализ, проектирование, код, тестирование и релизы требуются ровно в той степени, чтобы доставить продукт пользователю и увидеть, что реально работает. Затем следует сделать выводы и на основе новых знаний определить дальнейшие действия по улучшению продукта.

Это один из основных принципов непрерывной доставки (рис. 4.3).

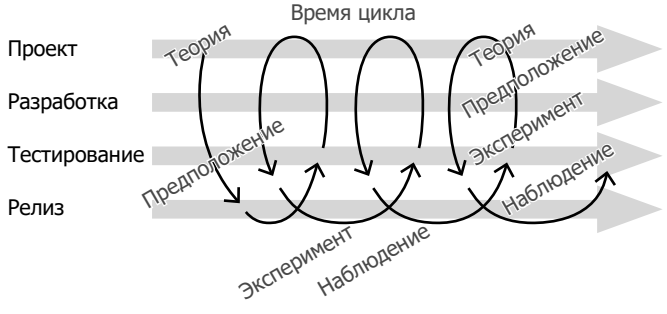

Рис. 4.3. Итерации непрерывной доставки

СИЛА ПЛАНИРОВАНИЯ

У сторонников водопадной разработки были благие намерения. Они считали, что это оптимальная стратегия успеха. В итоге мы десятилетиями пытались его добиться, но не смогли.

Сложность в том, что принципы водопада кажутся довольно разумными: «Тщательно обдумать, прежде чем начинать работу», «Грамотно спланировать действия и прилежно следовать этому плану». Исходя из опыта нашей отрасли, все это имеет смысл. Если ваш процесс строго регламентирован, подобный стиль даст отличные результаты.

При решении конкретных задач проблемы производственной инженерии и масштабирования обычно перевешивают проблемы дизайна. Однако сейчас это не так даже в промышленном производстве. По мере того как оно становится все более гибким и многие предприятия меняют

направленность, трансформируются даже самые жесткие производственные процессы. Производственный подход, тем не менее, преобладал в большинстве компаний на протяжении как минимум столетия, и мы сейчас остаемся его заложниками.

Чтобы осознать, что парадигма, в которой вы работаете, неверна, требуется огромное интеллектуальное напряжение. И оно кратно увеличивается, если осознать ошибку должен целый мир.

БИТВА ПРОЦЕССОВ

Если увеличение на порядок невозможно осуществить с помощью языка, формализации или построения диаграмм, то где искать выход?

И здесь чрезвычайно перспективно выглядит поиск по типичным для нашей отрасли способам самоорганизации и подходам к навыкам и техникам познания и открытий.

На заре программирования разработчики обычно имели математическое, естественно-научное или инженерное образование. Они действовали в одиночку или в небольших группах. Как большинство исследователей, они опирались на свой опыт и свои установки. Разработка на ранних этапах во многом напоминала математику.

Когда произошла компьютерная революция, спрос на разработку быстро превысил предложение. Требовалось как можно больше программных продуктов и как можно быстрее. Поэтому мы обратились к опыту других отраслей, чтобы научиться работать эффективно в масштабируемой среде.

И вот тут мы совершили ужасную ошибку, неверно истолковав саму природу разработки программных продуктов и заимствовав производственные техники. Мы привлекли целые армии программистов и попытались наладить аналог конвейерного производства товаров массового потребления.

Люди, которые совершили эту ошибку, не были глупыми, они просто заблуждались. Проблема была разноплановой. Программный продукт — комплексная вещь, и процесс ее создания не похож на привычную задачу производства, каким его видит большинство людей.

Первые попытки индустриализации разработки ПО были всеобъемлющими, болезненными и нанесли большой вред. Они привели к созданию множества некачественных продуктов. Решения оказались медленными, неэффективными, несвоевременными, они не соответствовали потребностям пользователей, и их было очень сложно обслуживать. В 1980–1990-е годы случился бум разработки и так же резко возросла сложность ее процессов во многих организациях.

Эта ошибка возникла, невзирая на тот факт, что в целом лидеры отрасли знали о существовании проблемы.

В книге Фреда Брукса *«Мифический человеко-месяц»*, изданной в 1970 году, как раз описана эта проблема и то, как ее избежать. Если вы еще не читали этот знаковый для нашей отрасли труд, вы, несомненно, удивитесь, как точно в нем сформулированы проблемы, с которыми вы сталкиваетесь практически ежедневно. И это несмотря на то, что Брукс описывает только свой опыт разработки ОС для суперкомпьютера IBM 360 в начале 1960-х годов, когда в его распоряжении имелись довольно примитивные технологии и ограниченный инструментарий. Тем не менее книга затрагивает гораздо более важные и ключевые понятия, чем язык, инструменты и технологии.

В то время многие команды создавали превосходные продукты, зачастую полностью игнорируя общепринятые способы планирования и управления проектами. Такие команды имели общие черты. Они были небольшими. Разработчики взаимодействовали с пользователями их продукта. Они внедряли идеи быстро и меняли курс, если эти идеи не работали, как ожидалось. Это был революционный подход – настолько революционный, что многие из этих команд, по сути, работали украдкой, поскольку их компании внедряли тяжеловесные процессы, которые тормозили их работу.

К концу 1990-х годов в ответ на эти тяжеловесные процессы многие разработчики начали искать более эффективные стратегии. Стали популярными несколько конкурирующих подходов к разработке – Crystal, Scrum, экстремальное программирование и другие. Их общий подход нашел отражение в agile-манифесте.

Революция, которую произвела agile-методология, коренным образом изменила принятую в разработке норму, но даже сегодня многие компании, если не сказать большинство, все еще ориентированы на планирование/водопадный подход.

Вдобавок ко всей сложности выявления проблемы такие организации иногда выдают желаемое за действительное. Очень хорошо, если организация способна:

- выявить действительные потребности пользователей;
- правильно определить, какую выгоду принесет организации удовлетворение этих потребностей;
- точно оценить затраты на удовлетворение этих потребностей;

- решить на основе рациональной оценки, превысит ли выгода эти затраты;
- составить грамотный план действий;
- четко придерживаться этого плана;
- посчитать затраты по окончании работы.

Проблема в том, что это не дает эффекта ни на уровне бизнеса, ни на уровне технологии. Реальный мир и разработка ПО как его составная часть так не работают.

Данные свидетельствуют, что две трети всех идей самых продвинутых мировых компаний-разработчиков на самом деле не приносят им прибыли либо приносят убыток[1]. Мы очень плохо понимаем, чего хотят пользователи. Даже если мы спросим их самих, они не ответят, потому что сами не знают этого. Самое эффективное здесь — итерация. Необходимо допустить, что некоторые, возможно, даже большинство наших идей — плохие, и испытывать их на практике как можно более быстрым, дешевым и эффективным способом.

Оценить ценность идеи для бизнеса тоже очень сложно, и это известный факт. Все знают слова главы IBM Томаса Дж. Уотсона (Thomas J. Watson), который предсказал, что мировая потребность в компьютерах составит целых пять штук!

Это не проблема технологий, это проблема человеческих ограничений. Чтобы двигаться вперед, следует использовать возможности, делать предположения, быть готовыми идти на риск. Но мы плохие прогнозисты. Поэтому, чтобы добиваться успеха, нам нужен такой способ самоорганизации, чтобы наши предположения не причинили нам вреда. Нам надо работать более осторожно, продвигаясь небольшими шагами и сужая радиус действия наших предположений, а также непрестанно обучаться. Следует работать итеративно!

Если у нас есть идея для работы, нам требуется способ определить, как завершить эту работу. Как остановить разработку плохой идеи? Если мы решили испытать решение, как сократить его масштаб так, чтобы не потерять вообще все, если оно окажется плохим? Нам необходимо уметь как можно быстрее выявлять плохие идеи. Здорово, если мы исключим

[1] Источник: Online Controlled Experiments at Large Scale, https://stanford.io/2LdjvmC.

их на этапе обдумывания. Но не все плохое так легко распознать. Успех — понятие скользкое. Идея может быть изначально хорошей, но несвоевременной, а ее исполнение — плохим.

Нам нужен быстрый и минимально затратный способ проверки предположений. Опрос, касающийся проектов в сфере разработки, проведенный в 2012 году McKinsey Group совместно с Оксфордским университетом, показал, что 17% крупных проектов (с бюджетом более 15 млн долларов США) были настолько плохи, что поставили под угрозу само существование компаний, которые их запустили. Как распознать такие плохие проекты? Если выполнять небольшие этапы, отслеживать прогресс и постоянно проверять и оценивать решение, мы скорее и с минимальными затратами заметим, если что-то пойдет не так, как задумывалось. При итеративном подходе стоимость каждого неверного шага несравнимо ниже, как и уровень риска.

В книге «Начало бесконечности» Дэвид Дойч описывает ключевое различие между идеями, применение которых ограниченно, и теми, для которых таких ограничений нет. Разница между плановым, водопадным, ориентированным на процесс подходом и итеративным, исследовательским и экспериментальным подобна разнице этих двух фундаментальных видов идей. Модели управления регламентированными процессами[1] требуют наличия регламентированного процесса. Согласно определению, этот процесс ограничен. Ограничителем служит в какой-то степени способность человеческого мозга удерживать подробности всего процесса. Наш интеллект позволяет применять абстракцию и модульность, чтобы скрыть некоторые ненужные подробности, но необходимость представить некий план всего процесса вынуждает нас удерживать в уме всю его последовательность. Это изначально ограниченный подход к решению задач. Применяя его, мы способны решать только те задачи, которые понятны заранее.

Итеративный подход заметно отличается. Мы можем приступить к работе, даже если почти ничего не знаем о задаче, и все равно добиться успеха. Мы

[1] Кен Швабер (Ken Schwaber) описывал водопадную модель как модель управления регламентированным процессом, которой он дал такое определение: «Модель управления регламентированным процессом требует полного понимания каждого этапа работы. Регламентированный процесс начинается и выполняется до своего окончания с одинаковым результатом». Швабер сравнивает эту модель с моделью управления эмпирическим процессом, которую использует agile-методология. См. https://bit.ly/2UiaZdS.

можем начать с той части системы, которая проста и понятна. Взять ее за основу исследования, пробовать кажущиеся подходящими архитектуру и технологии и так далее. Ничто здесь не является строго определенным. Мы получим результат, даже если поймем, что выбрали неподходящую технологию или архитектуру. Мы получим новые знания. Это изначально открытый, бесконечный процесс. Имея в своем распоряжении нечто вроде фитнес-функции, которая подсказывает нам, приближаемся ли мы к цели или отдаляемся от нее, мы можем продолжать действовать бесконечно, уточняя, расширяя и улучшая свои представления, идеи, навыки и продукты. Мы даже можем изменить эту фитнес-функцию, если поймем, что способны достичь лучших целей.

НАЧАЛО БЕСКОНЕЧНОСТИ

В необычайно познавательной книге «Начало бесконечности» физик Дэвид Дойч определяет науку и просвещение как поиск хороших объяснений и показывает, как на протяжении истории развития человечества разные идеи формируют «начало бесконечности», благодаря которому мы находим понятное и подходящее применение этим хорошим объяснениям.

Отличный тому пример – различие между алфавитным и пиктографическим письмом.

Первой формой письма была пиктография, и она до сих пор используется в китайской и японской письменности. Она очень красива, но имеет серьезный недостаток. Если вы услышите незнакомое слово, вы не сможете его записать, пока кто-то не научит вас. Пиктографическое письмо не инкрементно. Вам придется выучить символ для каждого слова. (В китайском языке примерно 50 000 иероглифов.)

Алфавит – совершенно иная система. Знаки алфавита обозначают звуки, а не слова. Вы сможете записать слово, возможно, с ошибками, но так, что любой поймет, что вы написали.

Вы сделаете это, даже если никогда не слышали этого слова или не видели, как оно пишется.

Точно так же вы сможете прочесть незнакомое слово. Вы сможете читать слова, не зная, как они произносятся или что означают. Это исключено при использовании пиктограмм. Таким образом, алфавитное письмо бесконечно, а пиктографическое ограниченно. Первое можно масштабировать для представления различных идей, второе – нет.

Такая бесконечность применения справедлива для agile-методологии и неверна для водопадной модели.

Водопадная разработка последовательна. Прежде чем перейти на следующий этап, вам необходимо решить вопросы текущего. Это означает, что каким бы интеллектом мы ни обладали, существует предел, когда сложность совокупной системы превосходит рамки человеческого понимания.

Возможности человеческого разума ограниченны, но способность человека понимать – не обязательно. Физиологические ограничения удается преодолеть с помощью технологий, которые мы развиваем и совершенствуем. Мы способны мыслить абстракциями и категориями (модульно) и таким образом расширять степень понимания.

Agile-методология построена на работе с небольшими фрагментами. Легко начать действовать, не имея ответов на все вопросы. Этот подход предполагает достижение успеха, возможно, не самыми лучшими и иногда даже неподходящими способами, но тем не менее благодаря ему на каждом этапе мы узнаем что-то новое.

Мы уточняем свою мысль, определяем следующий этап и затем реализуем его. Agile-разработка безгранична и бесконечна, поскольку мы каждый раз работаем с небольшой областью и двигаемся вперед с уже изученной и понятной позиции. Это более органичный и эволюционный подход к решению задачи.

Именно поэтому agile представляет собой важный шаг вперед как способ решения сложных задач.

Это не значит, что методология agile идеальна или дает все ответы. Но она очень важна для достижения лучшей производительности.

Сила планирования — кажущаяся. Оно не помогает добиться большей точности, управляемости и профессионализма. Это скорее ограничивающий подход, основанный на предположениях и работающий только в небольших, понятных системах с четкой структурой.

Из этого следуют очень важные выводы. Мы должны, говоря словами Кента Бека, которые он использовал как подзаголовок к своей классической книге *«Экстремальное программирование»*, «объять изменения»!

Нам необходимо иметь уверенность, когда мы приступаем к работе, не зная ответов и не имея точного понятия об объемах этой работы. Это внушает опасение многим компаниям и специалистам, но на самом деле люди почти всегда ведут себя именно таким образом. Когда кто-то открывает новый бизнес, он не может знать точно, когда к нему придет успех, и придет ли вообще. Ему неизвестно, скольким людям придется по душе новый продукт и будут ли они готовы заплатить за него.