ГЛАВА 1
Что есть Agile

Agile везде. И, как ни парадоксально, нигде.

Двадцать лет назад локомотив Agile с ревом ворвался в сознание разработчиков программного обеспечения. За это время количество компаний, называющих себя Agile, возросло на порядки. А сколько команд *действительно* применяет Agile-подход в своей работе? Не так много. Agile, легко повторяемое название, пользуется огромным успехом. А как насчет идей, стоящих за названием? Вообще-то большинство из *них* игнорируется.

Исправим это.

ПРОИСХОЖДЕНИЕ AGILE

В 1990-е считалось, что разработка ПО находится в кризисе. Это явление так и называлось: «Кризис программного обеспечения». Проекты разработки выходили за рамки бюджетов, задерживались, не отвечали требованиям, и (согласно часто цитируемому и зловеще именуемому «Отчету о Хаосе» (CHAOS Report)) почти треть их была полностью отменена [Standish1994].

Agile не был ответом на этот кризис. Отнюдь нет. Agile стал *ответным подходом*.

Чтобы взять под контроль разработку ПО, большие организации придумали высокодетализированный процесс, точно определявший, как ПО должно создаваться. Все жестко контролировалось, чтобы исключить возможность ошибки. (Во всяком случае, в теории.)

Сначала бизнес-аналитики должны были опрашивать стейкхолдеров (stakeholders, заинтересованные стороны) и документировать системные требования. Далее архитекторы программного обеспечения должны были изучить документы с требованиями и создать подробную проектную документацию, определяющую каждый компонент системы и их взаимосвязь друг с другом. Затем программисты должны были конвертировать проектную документацию в код. В некоторых организациях такое выполнение механического перевода считалось низкоквалифицированной работой.

Тем временем руководители тестирования должны были создавать планы тестирования, используя те же документы, и когда кодирование завершалось, бесчисленные специалисты по обеспечению качества должны были вручную выполнять эти планы, фиксируя отклонения как дефекты. По завершении каждой фазы все должно было быть задокументировано, проверено и подписано.

Подход, основанный на фазах, получил название водопадной (waterfall development) или каскадной разработки (phase-gate development)[1]. Если вам это кажется похожим на нелепое пугало — считайте, что вам повезло. Не все команды применяли в 1990-х перегруженный документацией каскадный процесс, но он широко признавался как логичный и разумный метод работы. Конечно, нужно было определить требования, разработать проект, затем выполнить реализацию и следом — тестирование. Конечно, нужно было документировать каждую фазу. Это была *дисциплина*. Это была *инженерия*. Как еще можно было добиться успеха?

РОЖДЕННЫЙ ИЗ КРИЗИСА

Крупные компании расписывали свои процессы в мельчайших деталях. Роли, ответственности, шаблоны документов, языки моделирования, советы по контролю за изменениями... каждый аспект разработки строго регламентировался и контролировался. Если проект заканчивался провалом (а согласно CHAOS Report, менее одной шестой доли проектов были успешны), причиной считалась недостаточная детализация процесса: нужно больше документов, больше согласований. Это порождало огромное количество документации. Мартин Фаулер описывал это в своей статье *The Almighty Thud* [Fowler1997].

Этот стиль работы был далеко не лучшим. Он был бюрократическим и бесчеловечным. Казалось, знания и навыки не так важны, как соблюдение процессов. Программисты чувствовали себя легко заменимыми винтиками бездушной машины. И даже она работала не очень хорошо.

И тогда несколько человек создали более простые, изящные и менее директивные методы разработки программного обеспечения. Они назывались легковесными в отличие от тяжеловесных, использовавшихся большими компаниями. Эти методы носили такие названия, как адаптивная разработка программного обеспечения (Adaptive Software Development), Crystal, разработка на основе функциональности (Feature-Driven Development, FDD), метод разработки динамических систем (Dynamic Systems Development Method, DSDM), экстремальное программирование (XP) и Scrum.

К концу 1990-х эти методы стали привлекать серьезное внимание. Экстремальное программирование, в частности, вызвало вспышку массового интереса среди программистов. В 2001 году 17 сторонников легковесной методологии встретились на горнолыжном курорте в штате Юта, чтобы обсудить объединение усилий.

МАНИФЕСТ AGILE

«Лично я не ожидал, что эта конкретная группа [людей] сможет договориться о чем-либо по существу», — позже говорил Алистер Кокберн.

И действительно, за два дня они смогли договориться только о двух вещах: названии Agile и заявлении о четырех ценностях новой методологии (рис. 1.1). В течение

[1] Источником появления подхода Waterfall часто ошибочно считают статью Уинстона Ройса, написанную в 1970 году. Однако применение фазового подхода восходит еще к 1950-м, а статья Ройса не пользовалась широкой известностью до конца 1980-х, когда к ней стали прибегать для описания того, что люди уже давно делали [Bossavit2013, chapt. 7].

следующих нескольких месяцев, переписываясь по электронной почте, эти люди сформулировали 12 сопутствующих принципов (рис. 1.2) [Beck2001].

Манифест гибкой разработки программного обеспечения Agile

Мы постоянно открываем для себя более совершенные методы разработки программного обеспечения, занимаясь разработкой непосредственно и помогая в этом другим. Благодаря проделанной работе мы смогли осознать, что:

— люди и взаимодействие важнее процессов и инструментов;
— работающий продукт важнее исчерпывающей документации;
— сотрудничество с заказчиком важнее согласования условий контракта;
— готовность к изменениям важнее следования первоначальному плану.

То есть, не отрицая важности того, что справа, мы все-таки больше ценим то, что слева.

Kent Beck	Martin Fowler	Ron Jeffries	Steve Mellor
Mike Beedle	James Grenning	Jon Kern	Ken Schwaber
Arie van Bennekum	Jim Highsmith	Brian Marick	Jeff Sutherland
Alistair Cockburn	Andrew Hunt	Robert C. Martin	Dave Thomas
Ward Cunningham			

Рис. 1.1. Ценности Agile

Основополагающие принципы Манифеста Agile

Наивысшим приоритетом для нас является удовлетворение потребностей заказчика благодаря регулярной и ранней поставке ценного программного обеспечения.

Изменение требований приветствуется даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества.

Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев.

На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.

Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.

Непосредственное общение является наиболее практичным и эффективным способом обмена информацией как с самой командой, так и внутри команды.

Работающий продукт — основной показатель прогресса.

Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм. Agile помогает наладить такой устойчивый процесс разработки.

Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.

Простота — искусство минимизации лишней работы — крайне необходима.

Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.

Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

Рис. 1.2. Принципы Agile

Это стало Манифестом Agile, который изменил мир. «Таким образом, — продолжил Алистер, — они в конце концов все же смогли договориться о чем-то по существу»[1].

Однако единого метода Agile не было. Его никогда не было и никогда не будет. Agile подразумевает три составляющие: название, ценности и принципы. И все. Это не что-то, что можно делать. Это философия. Это способ мышления, посвященный разработке программного обеспечения. Невозможно «использовать» Agile или «делать» Agile... вы только можете *быть* Agile. Или не быть. Если ваши команды воплощают философию Agile, то вы — Agile. Если не воплощают — то нет.

Если ваши команды воплощают философию Agile, то вы — Agile. Если не воплощают — то нет.

СУТЬ AGILE

Мартин Фаулер сделал карьеру, превращая сложные вопросы о программировании в тщательно продуманные, беспристрастно точные объяснения. Его определение «Суть разработки программного обеспечения по Agile» — одно из лучших:

> Agile-разработка является скорее *адаптивной*, чем предиктивной; *ориентированной* скорее *на людей*, чем на процессы[2].
>
> *Мартин Фаулер*

Адаптивность вместо предиктивности

Помните, в CHAOS Report утверждалось, что всего одна шестая часть проектов в области программного обеспечения успешна? В отчете успешность определена очень специфическим образом:

- *успешный* — проект выполнен вовремя и в рамках бюджета, все характеристики и функциональности соответствуют изначально запланированным;
- *сомнительный* — проект завершен, результаты переданы в эксплуатацию, но количество характеристик и функциональностей меньше, чем было заявлено изначально. Запланированные бюджет и сроки проекта превышены;
- *неуспешный* — проект отменен в какой-либо момент в течение цикла разработки.

---

[1] Алистер Кокберн, цитата Джима Хайсмита в [Highsmith2001]. Полная цитата: «Лично я не ожидал... что эта конкретная группа приверженцев различных гибких методик сможет договориться о чем-либо по существу... Что касается меня, я в восторге от окончательной формулировки [Манифеста]. Я был удивлен, что и другие оказались так же довольны окончательной формулировкой. Таким образом, мы все же смогли договориться о чем-то по существу».

[2] Фаулер выражал эту идею множество раз в течение нескольких лет. Оригинальную версию можно найти в статье [Fowler2000].

Эти определения прекрасно иллюстрируют предиктивный тип мышления. Они все говорят о *соответствии запланированному*. Если вы сделали то, что собирались сделать, то вы успешны. А если нет, то неуспешны! Все просто.

На первый взгляд, это разумно. Но посмотрим повнимательнее. Тут чего-то не хватает. Райан Нельсон писал в журнале *CIO Magazine* [Nelson2006]:

> Как обнаружилось, проекты, отвечающие всем традиционным критериям успешности — время, бюджет и технические спецификации, — могут по завершении все же быть провальными, поскольку оказываются неактуальными для пользователей, которым предназначались, или потому, что в итоге не приносят особых выгод бизнесу... С другой стороны, проекты, считающиеся неуспешными согласно традиционным метрикам информационных технологий, могут оказаться успешными, поскольку, несмотря на проблемы с бюджетом, графиком или техническими характеристиками, получившаяся система нравится конечным пользователям или имеет какую-либо непредвиденную ценность.

Команды Agile определяют успех как *поставку ценности*, а не соответствие плану. Фактически команды, действительно достигнувшие Agile, активно ищут возможности повысить ценность продукта, *изменив* планы.

Команды Agile определяют успех как поставку ценности, а не соответствие плану.

Взгляните еще раз на Манифест (см. рис. 1.1 и 1.2). Найдите пару минут, чтобы вдумчиво изучить ценности и принципы Agile. Сколько из них относятся к поставкам ценного программного обеспечения и адаптации к полученной обратной связи?

Ориентированность на людей, а не на процессы

В случае тяжеловесных процессов люди пытались избежать ошибок, тщательно определяя каждый аспект разработки ПО. При добавлении в процессы регламента индивидуальные навыки становились менее важными. В теории вы могли применять один и тот же процесс снова и снова с разными людьми и получать одни и те же результаты. (Если вдуматься, то они и получали. Просто не те результаты, которые хотели.)

Agile заявляет, что люди — главный фактор успеха в разработке программного обеспечения. Не только их знания и навыки, но и все аспекты человеческой природы. Насколько хорошо сработались члены команды. Насколько часто они отвлекаются. Насколько комфортно и безопасно для них высказывать свое мнение. Насколько они увлечены своей работой.

Agile заявляет, что люди — главный фактор успеха в разработке программного обеспечения.

У Agile-команд тоже есть процессы (у любой команды они есть, пусть и неявные), но они находятся на службе у человека, а не наоборот. И Agile-команды сами отвечают за свои процессы. Желая улучшить методы работы, люди меняют процессы.

Посмотрите на Манифест еще раз (см. рис. 1.1 и 1.2). Какие ценности и принципы имеют отношение к концепции «люди на первом месте»?

ПОЧЕМУ AGILE ПОБЕДИЛ

В течение первых десяти лет после появления Манифеста Agile столкнулся с небывалой критикой. Его называли «недисциплинированным». Говорили, что он никогда не будет работать. Следующие десять лет критики молчали. Agile был уже везде, по крайней мере это название. Тяжеловесные водопадные методы практически умерли. Молодые программисты вообще не верили в то, что кто-то когда-то мог работать таким образом.

Не то чтобы основанные на фазах процессы неполноценны по сути. У них, безусловно, есть свои недостатки, но если ограничивать их объем, при этом действуя в хорошо изученной предметной области, то методы водопадного стиля тоже могут работать. Проблема была в самом тяжеловесном процессе, который применяли крупные компании. Словно по иронии судьбы, процессы, предназначенные для того, чтобы *избегать* проблем, на самом деле сами *вызывали* большинство проблем, с которыми сталкивались компании.

Получение информации и реакция на изменения лежат в основе всего того, что называется Agile.

Трудно представить, как будет работать программа, до того, как вы начнете ее использовать на практике, и еще тяжелее продумать абсолютно все, что она должна будет делать. И это вдвойне сложнее для людей, которые не вовлечены в разработку программного обеспечения. В результате критически важно как можно скорее предоставить людям работающую программу. Вам просто необходимо получить от них обратную связь о том, что не работает и чего не хватает, и затем скорректировать ваши планы в зависимости от полученной информации. Как говорится в Манифесте, «работающее программное обеспечение — основной показатель прогресса». Получение информации и реакция на изменения лежат в основе всего того, что называется Agile.

В случае тяжеловесных процессов придавалось настолько большое значение контролю над процессами и согласованию документации, что порождались значительные задержки и расходы. На то, чтобы получить работающее ПО, уходили годы, и заказчику не показывали ничего конкретного почти до самого конца проекта. Вместо того чтобы приветствовать изменения, в этих процессах делалось все, чтобы их *избежать*. Была даже отдельная составная часть процессов «Совет по контролю за изменениями» (Change Control Board), чьей основной задачей было сказать «нет» запросам на изменения. (Точнее, «да, но за это понадобится доплатить».)

Все это наслаивалось друг на друга в проектах, где люди годами вели разработку, прежде чем могли что-то показать клиенту. И когда они это делали, было уже слишком поздно и дорого что-то менять. В конечном итоге они выдавали программное обеспечение, которое не делало то, чего хотел заказчик.

Типичный провал тяжеловесного процесса

Третьего февраля 2005 года Роберт С. Мюллер III, директор Федерального бюро расследований, предстал перед подкомитетом Сената США, чтобы объяснить, как ФБР умудрилось потратить впустую 104,5 миллиона долларов[1].

Это явно было не комфортно для компании. В июне 2001-го ФБР запустило проект VCF с целью заменить программное обеспечение для управления делами. Через четыре года он был отменен. Общие затраты составили 170 миллионов долларов; 104,5 миллиона из них были полностью невозвратными.

Сроки и последовательность событий проекта могут рассказать нам хорошо знакомую историю. Проект начался в июне 2001 года. Через 17 месяцев, в ноябре 2002-го, были определены «четкие требования». Программное обеспечение было поставлено год спустя, в декабре 2003-го, но «ФБР сразу обнаружило некоторое количество недостатков в VCF, которые делали его непригодным для использования». Подрядчик, разрабатывавший программу, согласился исправить недостатки, но только за дополнительную плату в размере 56 миллионов долларов и в срок один год. В конце концов ФБР отказалось от идеи исправлять недочеты, фактически перечеркнув годы работы.

Agile-команды демонстрируют прогресс, используя работающие программы, а не документы.

Несмотря на существование большого разнообразия подходов к Agile (и некоторые из них больше используют популярное название, чем действительно следуют философии), у них есть кое-что общее. Они фокусируются на том, чтобы делать прогресс в работе видимым, позволяя всем стейкхолдерам проекта корректировать его на ходу. Казалось бы, это мелочь, но она невероятно эффективна. Именно благодаря ей мы больше не слышим о кризисе ПО. Сроки разработки программ все еще задерживаются. Бюджеты все еще перерасходуются. *Но Agile-команды демонстрируют прогресс, используя работающие программы, а не документы.* С самого начала. И это огромное достижение.

Agile имеет и другие преимущества, помимо наглядности процесса. Но даже одного этого оказалось достаточно, чтобы все захотели стать Agile.

ПОЧЕМУ AGILE РАБОТАЕТ

Первым прорывным успехом Agile стало экстремальное программирование, метод со слоганом «Примите изменения». В нем смешались здоровая доза философских размышлений о разработке программ и прагматичное стремление изменить ситуацию

[1] Источники: Mueller's February 3, 2005, testimony to Congress (https://oreil.ly/GlQSa) и Inspector General Glenn Fine's May 2, 2005, testimony to Congress (https://oig.justice.gov/node/672).

к лучшему. В предисловии к первой книге по экстремальному программированию было сказано:

> «Если коротко, то экстремальное программирование обещает снизить риски проекта, улучшить отклик на потребности бизнеса и повысить продуктивность в течение всего срока жизни системы, сделать приятным создание ПО в команде — и все это одновременно. Это правда. Прекратите смеяться» [Beck2000a].
>
> *Extreme Programming Explained, первое издание*

Одни действительно смеялись. А другие попробовали и обнаружили, что (вопреки всеобщему мнению о том, как должна вестись разработка ПО) экстремальное программирование реально выполняло все, что обещало. Так, несмотря на смех, оно пользовалось спросом, а вместе с ним Agile.

Экстремальное программирование было живым примером Agile, заложившим основу идей и терминологии, которые используются до сих пор. Однако сила сообщества Agile в том, что оно всегда было широкой коалицией. Agile не ограничивается каким-то одним методом. Он постоянно расширяется, включая в себя новых людей и свежие идеи. «Бережливая разработка программного обеспечения» (Lean software development), Scrum, Kanban, «Бережливый стартап» (Lean Startup), DevOps и многие-многие другие подходы внесли свой вклад в то, что сейчас известно под общим названием Agile.

Если взять все эти идеи и сгруппировать по категориям, то получится пять центральных концепций.

- *Полагайтесь на людей.* Создавайте процессы, которые понятны и могут работать с учетом человеческой природы. Отдайте право принимать решения в руки тех, кто наиболее квалифицирован в этой области. Выстраивайте здоровые рабочие взаимоотношения, основанные на сотрудничестве.
- *Поставляйте полезный продукт.* Запрашивайте обратную связь, экспериментируйте и подстраивайте свои планы. Считайте частично выполненную работу затратами, а не прибылью. Концентрируйтесь на создании ценных результатов. Поставляйте продукт часто.
- *Исключите напрасную трату времени и усилий.* Выполняйте работу маленькими обратимыми шагами. Примите возможность неудачи и стройте ваши планы с учетом вероятности их быстрого провала. Максимизируйте невыполненную работу. Стремитесь к производительности, а не эффективности.
- *Стремитесь к техническому совершенству.* Обеспечивайте гибкость с помощью технического качества. Закладывайте в проект то, что известно, а не то, что предполагаете. Начинайте с простого и добавляйте сложное, только если это будет реально необходимо. Создавайте системы, которые легко развивать, даже (и особенно) в непредвиденных направлениях.
- *Совершенствуйте ваши процессы.* Экспериментируйте с новыми идеями. Настраивайте и адаптируйте то, что работает. Никогда не считайте, что отлаженный, популярный способ будет самым лучшим и для вас.