Канбан

АЛЬТЕР-НАТИВНЫЙ ПУТЬ В AGILE

ДЭВИД АНДЕРСОН

СОДЕРЖАНИЕ

Часть III. Внедрение Канбана

ГЛАВА 1

РЕШЕНИЕ ДИЛЕММЫ AGILE-МЕНЕДЖЕРА

В 2002 году я был менеджером по разработке в удаленном офисе подразделения мобильных телефонов Motorola в Сиэтле (оно называлось PCS) и оказался в сложной ситуации. Мой отдел был частью стартапа, который Motorola приобрела годом ранее. Мы разрабатывали сетевое ПО для беспроводной передачи данных, например беспроводного скачивания и управления приборами. Эти серверные приложения входили в интегрированные системы, которые были тесно связаны с клиентским кодом мобильных телефонов, а также с другими элементами в телекоммуникационных сетях и операционной инфраструктуре, например с биллингом. Дедлайны назначались менеджерами, которые не обращали внимания на инженерную сложность проекта, его риски или масштаб. Основа кода развивалась из стартапа, при этом разработка многих первоначально запланированных возможностей была отложен на потом. Один старший разработчик настаивал на том, чтобы наши продукты назывались «прототипами». Нам было отчаянно необходимо повысить производительность и качество продукции, чтобы соответствовать требованиям бизнеса.

В своей повседневной деятельности в 2002 году и в процессе работы над предыдущей книгой[1] я был обеспокоен в основном двумя проблемами. Во-первых, как защитить команду от постоянно растущих требований бизнеса и достичь того, что сейчас в agile-сообществе принято

называть «оптимальным темпом». И во-вторых, как я могу успешно внедрить agile-подход в масштабах всей организации, преодолев неизбежное сопротивление переменам?

В ПОИСКАХ ОПТИМАЛЬНОГО ТЕМПА

В 2002 году agile-сообщество воспринимало оптимальный темп просто как «40-часовую рабочую неделю»[2]. Принципы Agile-манифеста[3] гласили, что «agile-процессы способствуют оптимальному развитию. Спонсоры, разработчики и пользователи должны быть готовы поддерживать постоянный темп в течение бесконечного времени». За два года до этого моя команда в Sprint PCS постоянно слышала от меня, что «масштабная разработка ПО — это марафон, а не спринт». Если членам команды предстояло поддерживать неизменный темп в процессе работы над полуторагодичным проектом, то нельзя было позволить им сгореть за месяц-другой. Проект нужно было распланировать, вставить в бюджет, расписать по времени и подвергнуть оценке, чтобы члены команды ежедневно тратили на работу разумное количество времени и не слишком уставали. Передо мной как менеджером стояла задача достичь этой цели *и* удовлетворить все требования бизнеса.

Когда я работал на первой менеджерской должности (это было в 1991 году, в стартапе, который делал платы захвата видео для персональных и более мелких компьютеров), CEO* сообщил, что у руководства сложилось обо мне «крайне отрицательное мнение». Я всегда отвечал «нет», когда наши возможности как разработчиков уже были исчерпаны, а от нас требовали все больше продуктов или функций. К 2002 году это вошло у меня в привычку: я провел еще десять лет, отказываясь выполнять капризы владельцев бизнеса.

* Chief Executive Officer — главный исполнительный директор (генеральный директор). *Прим. ред.*

Команды разработчиков и IT-отделы компаний сильно зависят от других групп, которые постоянно торгуются, упрашивают, угрожают и переделывают даже самые очевидные и объективно разработанные планы. В число уязвимых попадают и планы, основанные на тщательном анализе и историческом опыте. Большинство же команд, не имевших тщательных методов анализа и исторического опыта, не могли совладать с теми, кто подталкивал их брать на себя непонятные, а нередко и неосуществимые обязательства.

Тем временем сотрудники смирились с безумной загрузкой, и непомерные объемы работы стали нормой. Кажется, никто не задумывался над тем, что у инженеров-программистов тоже может быть общественная или семейная жизнь. Звучит грубо, но это правда! Я знаю слишком много примеров, когда излишняя производственная нагрузка навсегда разрушила семейные отношения. Трудно сочувствовать типичному гику-разработчику. В моем родном штате Вашингтон доход инженера-программиста уступает только заработку стоматолога. Как и во времена Форда, то есть в 1920-е годы, когда люди на его заводах зарабатывали в пять раз больше, чем в среднем по стране, никому и в голову не приходит подумать о монотонности работы или о благополучии специалистов: им столько платят! Трудно представить себе наличие профсоюзов в таких интеллектуальных отраслях, как разработка ПО, потому что никто не станет всерьез изучать причины физических и психологических недугов, которые испытывают разработчики. Более ответственные работодатели предлагают, например, такие меры, как массаж или психотерапия. Или проводят дни психического здоровья — и это вместо того, чтобы уделить внимание изучению основных причин проблемы. Технический писатель, сотрудник известной фирмы — разработчика ПО, однажды сказал мне: «Нет ничего страшного в том, что я употребляю антидепрессанты, ведь так поступают все!» Программисты обычно соглашаются со всеми требованиями, получают неплохую зарплату и страдают от последствий. Я хотел изменить такое положение дел, найти взаимовыгодный

подход, который позволял бы мне говорить «да» и при этом все же защищать свою команду, обеспечивая достижение оптимального темпа. Я хотел вернуть членов своей команды в общество и семью и улучшить условия, которые вызывали у разработчиков, не достигших и тридцати лет, стресс и проблемы со здоровьем. Поэтому я решил начать бороться с этими проблемами.

В ПОИСКАХ УСПЕШНОГО УПРАВЛЕНИЯ ИЗМЕНЕНИЯМИ

Вторая проблема, которая занимала мои мысли, — это управление изменениями в крупных организациях. Я был менеджером по разработке в Sprint PCS и затем в Motorola. В обеих компаниях существовала серьезная потребность в переходе на более гибкие методы работы. Но в обоих случаях у меня возникали трудности при внедрении agile-методов более чем в одной-двух командах.

Оба раза у меня не было достаточно полномочий, чтобы просто приказать внести изменения в работу множества команд. Я старался внедрить изменения по просьбе высшего руководства, но не обладал должной властью. Меня просили повлиять на коллег, чтобы те внедрили в своих командах такие же изменения, как я — в своей. Но они не торопились брать на вооружение методы, которые, казалось бы, зарекомендовали себя в моей команде наилучшим образом. У этого сопротивления, вероятно, было несколько причин. Чаще всего я слышал, что у каждой команды своя ситуация и мои методы нужно будет подгонять под конкретные нужды других. К середине 2002 года я понял, что жестко предписывать какой-либо процесс разработки ПО бесполезно — это просто не будет работать.

Процесс нужно было адаптировать для каждой конкретной ситуации, поэтому требовалось активное руководство каждой командой. А этого нередко не хватало. Даже при должном руководстве нет полной уверенности

в том, что существенные изменения могут произойти без наличия установленной структуры и советов по поводу того, как подогнать процессы под иные ситуации. Если у руководителя, коуча или ответственного инженера нет четкого представления о том, что делать, то любая адаптация, скорее всего, пройдет субъективно. При этом велика вероятность ошибок — например, внедрения неподходящего шаблона процесса.

Я попытался осветить эту проблему в книге Agile Management for Software Engineering, которую писал в то время. Я задавался вопросом: «Почему гибкая разработка дает лучшие экономические результаты, чем традиционные подходы?» Я хотел применить с этой целью структуру теории ограничений[4].

В процессе исследований и написания упомянутой книги я понял, что уникальна каждая ситуация. Да и разве может сдерживающий фактор или узкое место оказаться одинаковым для любой команды и проекта в любое время? Каждая команда уникальна: иные навыки, возможности, опыт. Каждый проект отличается от других бюджетом, расписанием, объемом и рисками. Непохожи друг на друга и организации: у них разные цепочки создания ценности, они работают на различных рынках (рис. 1.1). Мне показалось, что это может дать ключ к пониманию сопротивления изменениям. Если предлагаемые перемены в методах работы и моделях поведения не дают, по мнению сотрудника, очевидного преимущества, то он не примет их. Если эти изменения не влияют на то, что воспринимается командой как ограничитель или сдерживающий фактор, то команда будет сопротивляться. Иными словами, перемены, предложенные без учета контекста, будут отвергнуты сотрудниками, которые прекрасно знают контекст работы.

Казалось бы, будет лучше, если новый процесс начнет развиваться, устраняя одно ограничение за другим. Это основное положение теории ограничений Голдратта. Понимая, что мне еще многому предстоит научиться, я осознавал ценность материала и устремился вперед в работе над рукописью. Мне было ясно, что книга не давала советов, как внедрить

идеи в более широком масштабе, а также почти не помогала найти способы управления изменениями.

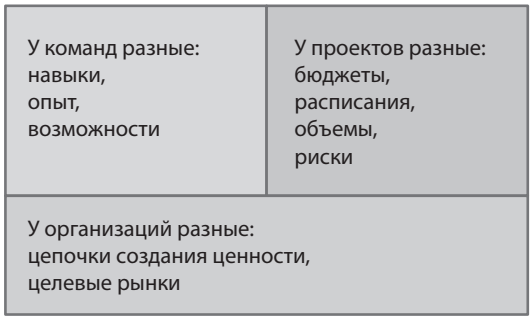

Рис. 1.1. Почему универсальные методологии разработки неверны

Подход Голдратта, изложенный в главе 16, направлен на поиск ограничений, а затем и способов их устранения, чтобы они перестали препятствовать производительности. После этого возникает новое ограничение, и цикл повторяется. Это итеративный подход, предполагающий систематическое улучшение производительности посредством выявления и устранения препятствий.

Я понял, что можно сочетать этот подход с некоторыми приемами из области бережливого производства. Смоделировав рабочий процесс жизненного цикла разработки ПО как потока создания ценности и создав систему отслеживания и визуализации для фиксации изменений состояния возникающей работы, «протекающей» по системе, я мог определить ограничители. Способность выявить ограничение — это первый шаг к модели, лежащей в основе TOC. Голдратт уже разработал применение этой теории для проблем рабочего процесса, носящее неуклюжее название «барабан-буфер-канат». Однако я понял, что упрощенный вариант этой системы можно внедрить в область разработки ПО.

С точки зрения происхождения «барабан-буфер-канат» — это пример класса решений, известных как вытягивающие системы. Как мы увидим

в главе 2, канбан тоже один из примеров такого рода систем. Побочный эффект вытягивающих систем состоит в том, что они ограничивают объем незавершенной работы до установленного заранее количества, препятствуя перегрузке сотрудников. К тому же полностью загруженными остаются только работники, напрямую сталкивающиеся с ограничением; у всех остальных должно оставаться свободное время. Я понял, что вытягивающие системы способны решить обе волновавшие меня проблемы. Вытягивающая система позволит мне внедрять пошаговые изменения, что (как я надеялся) существенно уменьшит сопротивление им, а также облегчит достижение оптимального темпа. Я принял решение перейти на систему «барабан-буфер-канат» при первой возможности. Мне хотелось поэкспериментировать с пошаговой эволюцией процесса и посмотреть, насколько она обеспечивает оптимальный темп и снижает сопротивление изменениям.

Такая возможность появилась осенью 2004 года в Microsoft, что подробно описано в главе 4 этой книги в анализе примера.

ОТ СИСТЕМЫ «БАРАБАН-БУФЕР-КАНАТ» К КАНБАНУ

Применение решения «барабан-буфер-канат» в Microsoft дало свои результаты. Сопротивление было слабым, производительность выросла более чем втрое, время опережения сократилось на 90%, а предсказуемость повысилась на 98%. Осенью 2005 года я сообщил о достигнутых результатах на конференции в Барселоне[5], а зимой 2006 года сделал еще один доклад. Моя работа привлекла внимание Дональда Рейнертсена, который специально приехал ко мне в офис в Редмонде. Он хотел убедить меня, что все готово к полному переходу на канбан.

Кан-бан — японское слово, которое дословно переводится как «сигнальная доска». В производстве такая доска используется для визуализации нарастающего темпа производства, что позволяет давать больше

продукции. Сотрудники на каждом этапе процесса не могут перейти к следующей фазе работы, пока посредством канбан-доски не будет дан соответствующий сигнал. Хотя я знал о существовании этого механизма, я не был убежден, что он полезен или даже жизнеспособен применительно к интеллектуальной работе, особенно к разработке ПО. Я понимал, что канбан обеспечивает оптимальный темп, но не знал о его хорошей репутации в качестве метода стимулирования пошагового улучшения процессов. Я не знал, что Тайити Оно, один из создателей производственной системы Toyota, говорил: «Два основных принципа системы производства Toyota — это “точно-в-срок” и автоматизация с участием человека, или автономизация. Для управления системой используется инструмент — это и есть канбан». Иными словами, канбан жизненно важен для процесса *кайдзен* («непрерывное улучшение»), применяемого в Toyota. Это механизм, который заставляет систему работать. Сейчас, имея за плечами пятилетний опыт, я принимаю это как абсолютную истину.

К счастью, Дон привел убедительный аргумент в пользу перехода с системы «барабан-буфер-канат» на канбан. Он звучал довольно эзотерически: система канбан обеспечивает более гладкий переход с простоя в узком месте, чем «барабан-буфер-канат». Однако понимание подобной отличительной особенности не так уж обязательно, чтобы читать и понимать эту книгу.

Вновь изучив реализованное в Microsoft решение, я понял, что если бы мы сразу воспринимали его как результат системы канбан, то итог был бы таким же. Мне показалось интересным, что два разных подхода ведут к одному и тому же результату. Итак, поскольку получался один и тот же процесс, я не чувствовал себя обязанным воспринимать его исключительно как результат внедрения системы «барабан-буфер-канат».

Я стал предпочитать этому сложному словосочетанию термин «канбан». Он используется в бережливом производстве (то же, что производственная система Toyota). Эта совокупность знаний получила гораздо большее распространение и признание, чем теория ограничений.

Канбан, несмотря на свое японское происхождение, менее метафоричен, чем барабан, буфер и канат, вместе взятые. Это слово легче произносить, объяснять и, как оказалось впоследствии, преподавать и внедрять. Вот оно и закрепилось.

ВОЗНИКНОВЕНИЕ КАНБАН-МЕТОДА

В сентябре 2006 года я ушел из Microsoft и возглавил отдел разработки ПО в Corbis — расположенной в центре Сиэтла частной компании по хранению базы фотографий и защите интеллектуальной собственности. Вдохновившись достигнутым в Microsoft, я решил внедрить вытягивающую систему канбан в Corbis. И здесь результаты были весьма успешными, они привели к развитию большинства концепций, представленных в этой книге. Это расширенный набор тех концепций — визуализация рабочего процесса, типы элемента потока операций, каденции, классы обслуживания, особая отчетность руководства и анализ операций, — которые определяют Канбан-метод.

В этой книге я описываю Канбан (с большой буквы) как метод эволюционных изменений, использующий вытягивающую систему канбан (с маленькой буквы), визуализацию и другие инструменты для катализации введения идей бережливого производства в технологические разработки и IT-операции. Это эволюционный и пошаговый процесс. Канбан позволяет достичь оптимизации процесса, связанной с контекстом, с минимальным сопротивлением изменениям и при этом сохранить оптимальный темп для всех вовлеченных в работу сотрудников.

ПРИНЯТИЕ КАНБАНА В СООБЩЕСТВЕ

В мае 2007 года Рик Гарбер и я представили первые результаты работы в Corbis на конференции по бережливой разработке новых продуктов

в Чикаго. Доклад слушали примерно 55 человек. Летом того же года на конференции Agile 2007 в Вашингтоне я предложил провести круглый стол по обсуждению системы канбан — на него пришло человек двадцать пять. Через два дня один из присутствовавших, Арло Бельши, произнес краткую речь, в которой рассказывал о своем методе «обнаженного планирования»[6]. Оказалось, что и другие компании внедряют у себя вытягивающие системы. Так, в Yahoo! была создана дискуссионная группа, которая быстро набрала сотню членов. А сейчас в нее входит уже более 1000 человек. Некоторые из участников моего круглого стола решили попробовать Канбан на своем рабочем месте — нередко с командами, которые безуспешно боролись со Scrum. Самые известные из моих ранних последователей — Карл Скотланд, Аарон Сандерс и Джо Арнольд, все из Yahoo!. Эта компания быстро внедрила Канбан в более чем десяток команд на трех континентах. Еще один заметный участник круглого стола, Кендзи Хиранабе, разрабатывал канбан-решения в Японии. Вскоре после конференции он написал на эту тему две статьи для InfoQ[7, 8], которые вызвали большой интерес. Осенью 2007 года Санджив Огастайн, автор Managing Agile Projects[9] и основатель Организации руководителей гибких проектов (Agile Project Leadership Network, APLN), посетил Corbis в Сиэтле и описал наши канбан-системы как «первый новый agile-метод, который я увидел за пять лет».

На следующий год на конференции Agile 2008 в Торонто состоялось шесть презентаций решений канбан разного формата. Одна из них была проведена Джошуа Кериевски из Industrial Logic — компании по обучению и консалтингу в области экстремального программирования. В ней он продемонстрировал, как пришел к похожим идеям по взятию на вооружение принципов экстремального программирования и их улучшению в контексте его бизнеса. В тот год Agile Alliance вручил приз Гордона Паска Арло Бельши и Кендзи Хиранабе за их вклад в agile-сообщество. Как я понял, он состоял в одном случае во внедрении Канбана, а в другом — в разработке и пропаганде на удивление схожих идей.

издательство
МАНН, ИВАНОВ И ФЕРБЕР
Максимально полезные книги