

Майк Кон

AGILE

ОЦЕНКА И ПЛАНИРОВАНИЕ ПРОЕКТОВ

Перевод с английского



альпина
ПАБЛИШЕР

Москва

2018

Об авторе

Майк Кон — основатель Mountain Goat Software, фирмы, занимающейся консалтингом в сфере управления процессами и проектами. Майк специализируется на помощи компаниям в применении agile-подхода с целью повышения эффективности. Он также является автором книги «Пользовательские истории: Гибкая разработка программного обеспечения» и книг по языкам программирования Java и C++. За спиной у Майка более чем 20-летний опыт работы руководителем в организациях разного размера, от стартапа до компании из списка Fortune 40. Его статьи можно найти в таких изданиях, как *Better Software*, *IEEE Computer*, *Cutter IT Journal*, *Software Test and Quality Engineering*, *Agile Times* и *C/C++ Users Journal*. Он часто выступает на отраслевых конференциях, является соучредителем организации Agile Alliance и входит в ее совет директоров. Майк — сертифицированный Scrum-мастер и тренер, член Компьютерного общества IEEE и Ассоциации компьютерной техники.

Для получения дополнительной информации обращайтесь на сайт www.mountaingoatsoftware.com или отправьте запрос по адресу mike@mountaingoatsoftware.com.

Предисловие

Куда бы я ни попадал в agile-мире, везде слышал одни и те же вопросы:

- Как подходить к планированию, когда работает большая команда?
- Каким должен быть размер итерации?
- Как представлять руководству данные о прогрессе в разработке?
- Как приоритизировать истории?
- Как получить представление о проекте в целом?

Настоящая книга дает ясные ответы не только на эти, но и на многие другие вопросы. Если вы руководитель проекта, исполнитель проекта, разработчик или директор, то найдете здесь инструменты, необходимые для оценки, планирования и управления agile-проектами практически любого масштаба.

Я знаю Майка Кона уже пять лет. Мы познакомились вскоре после подписания Agile-манифеста. Майк привнес в организацию Agile Alliance заряд энтузиазма и энергии. Любой проект, за который бы он ни взялся, выполнялся полностью и на хорошем уровне. Его вклад был видимым и ценным. Майк очень быстро стал незаменимым членом этой молодой организации.

К созданию данной книги он подошел так же профессионально, тщательно и энергично. Этого нельзя не заметить, это сквозит в каждой строчке.

Это видно по тому, что рекомендации в книге носят исключительно практический характер. Здесь нет теоретических абстракций. Автор не заставляет читателя витать в облаках, созерцая проблемы с высоты 10 000 метров. Майк приводит конкретные примеры, методы, инструменты, графики, рецепты, а главное — аргументированные рекомендации. Эта книга — практическое руководство по оценке и планированию.

Она сдобрена описанием случаев из практики Майка, связанных с применением рассматриваемых методов и инструментов. Он рассказывает, когда они давали результат, а когда нет, говорит о том, где можно оступиться, а где все будет нормально. Вы найдете в книге не обещания, панацеи и гарантии, а в буквальном смысле кладезь добытого с большим трудом опыта.

Гибкий подход к оценке и планированию затрагивают многие работы,

однако таких, где этот подход является главной темой, единицы. Равных же этой книге по глубине и полезности вообще нет. В ней данная тема рассматривается настолько полно, настолько практично, что, на мой взгляд, ее можно считать фундаментальным трудом.

Возможно, я преувеличиваю, но она меня зацепила. Зацепила тем, что в ней даны квалифицированные ответы на многие давно ожидающие решения вопросы. Она стала инструментом, который я могу предложить клиентам, когда они ставят сложные проблемы. Меня радует, что эта книга вышла в свет и вот-вот попадет к читателю. Мне очень приятно, что она вошла в курируемую мной серию. Думаю, это бестселлер.

Роберт Мартин, редактор серии

Предисловие

Читая книгу или рукопись в первый раз, я всегда задаю себе вопрос: «Что нового автор привносит в эту область?» В случае Майка у меня два ответа: его книга расширяет наше представление о том, «как» подходить к оценке и планированию, а также «почему» важны определенные методы.

Agile-подход к планированию обманчив. Может показаться, что он довольно легок: создайте карточки историй, присвойте им приоритет, распределите их по итерациям, а затем добавьте детали — и получите план следующей итерации. Основы планирования можно объяснить команде за пару часов, и ей потребуется всего несколько часов, чтобы составить сносный план (для небольшого проекта). Книга Майка здорово помогает перейти от сносных планов к составлению очень хороших планов. В этом месте я предельно аккуратно подбираю слова. Я не говорю «превосходных планов», поскольку, как подчеркивает Майк в своей книге, выигрыш от превосходного плана по сравнению с (довольно) хорошим планом чаще всего оказывается не настолько значительным, чтобы тратить на него дополнительные силы.

Поначалу мои мысли о книге Майка крутились вокруг концепции agile-подхода к планированию. Меня всегда удивляло, а порой расстраивало массовое непонимание сути agile-подхода к планированию. То и дело слышишь саркастические замечания о том, что «команды agile-проектов не занимаются планированием» или «agile-команды не придерживаются ни определенных сроков, ни определенных требований». Даже Барри Боэм и Ричард Тернер не совсем правы в своей книге «Баланс гибкости и дисциплины: Руководство для сбитых с толку» (Balancing Agile and Discipline: A Guide for the Perplexed. Addison-Wesley, 2004), когда сравнивают традиционные методы планирования с agile-методами. Фактически Боэм и Тернер правильно понимают идею, но используют неверную терминологию. У них под термином «методы планирования» понимается «тщательно взвешенное сочетание прогноза и адаптивного приближения к прогнозу», а по отношению к термину «agile-методы» взвешивание является антонимом. Таким образом, противопоставление «традиционных методов планирования» и «agile-методов» несет совершенно неправильный посыл, что agile-команды не занимаются планированием. Нет ничего более далекого от реальной практики. Книга Майка дает

правильную установку — планирование является неотъемлемой частью любого agile-проекта. В ней очень много говорится о том, почему планирование так важно, и о том, как сделать его эффективным.

Во-первых, agile-команды планируют очень многое, но этот процесс более равномерно распределен по всему проекту. Во-вторых, agile-команды прямо учитывают тот критический фактор, который упускают из виду многие не использующие agile-подход команды, — неопределенность. Важно ли планирование? Несомненно, важно. Важна ли корректировка плана по мере накопления знаний и уменьшения неопределенности? Исключительно важна. Мне известно множество случаев, когда организации, которые на начальном этапе принимали нереальные обязательства, а потом не могли их выполнить, оказывались подходящими для заказчика, в то время как на тех, которые старались быть реалистичными (и понимали неопределенность), навешивали ярлык «неспособные соблюдать программу» или «некомандные игроки». Похоже, срыв поставки продукта считается приемлемым, а отказ принять обязательства (даже когда они очевидно нелепы) — нет. Agile-подход в мастерском представлении Майка сфокусирован на поставке ценного для пользователя продукта, а не на составлении ничем не оправданных и невыполнимых планов и принятии обязательств. Agile-разработчики, по существу, говорят: «Мы предоставим вам план на основе того, что нам известно в настоящий момент; мы будем адаптировать этот план так, чтобы реализовать вашу наиболее важную цель; мы будем адаптировать проект и наши планы по мере продвижения вперед и получения новой информации; мы ожидаем, что вы понимаете, о чем просите нас. Иными словами, гибкость, допускающая адаптацию к меняющимся условиям, и жесткое соблюдение первоначальных планов являются взаимоисключающими целями. В настоящей книге разбираются все эти утверждения.

Возвращаясь к критически важному вопросу управления неопределенностью, Майк превосходно показывает, как agile-подход к процессу разработки снижает одновременно и неопределенность целей (что мы реально хотим создать), и неопределенность средств их достижения (как мы будем создавать это). Многие сторонники традиционного планирования не понимают ключевого момента: планирование не устраняет неопределенность. Планы строятся на основе того, что мы знаем в данный момент. Неопределенность — это способ представления того, что нам неизвестно относительно целей и средств их реализации. Для большинства неопределенностей (отсутствия знания) единственным путем их снижения и приобретения знания является действие — выполнение каких-либо работ, создание чего-либо, моделирование чего-либо — и получение обратной связи. Подход многих руководителей проектов можно представить как «планирование, планирование, планирование — выполнение». Agile-подход

— это «планирование–выполнение–адаптация», «планирование–выполнение–адаптация». Чем выше неопределенности проекта, тем важнее применение agile-подхода для успеха.

Я бы хотел проиллюстрировать «как» и «почему» из книги Майка на примере глав 4 и 5, где детально показано, как оценивать пользовательские истории в пунктах или идеальных днях, а также приведены все за и против для каждого из этих подходов. Я практиковал оба подхода при работе с клиентами, но слова Майка помогли кристаллизироваться моим представлениям об оценке историй в пунктах и позволили понять, что пункты являются частью эволюции — эволюции в направлении простоты. Организации, занимающиеся разработкой программного обеспечения, давно ищут ответ на вопрос «насколько велик данный элемент программного обеспечения?». Строитель способен дать довольно обоснованную оценку, имея данные о площади здания. Оценки разных строителей могут варьировать, но размер фиксирован (хотя отделочные работы, требования к материалам и т.п. также влияют на оценку) и остается постоянным. Разработчики программного обеспечения давно хотят иметь подобный показатель.

В сфере разработки программного обеспечения для измерения размера продукта поначалу использовали количество строк программы (этот показатель до сих пор не вышел из употребления). В текущем планировании, однако, количество строк программы находит ограниченное применение по целому ряду причин, включая трудозатраты на их подсчет. Затем на сцену вышли функциональные точки (и несколько аналогичных идей). Функциональные точки устраняли некоторые проблемы показателя количества строк, но по-прежнему требовали значительных трудозатрат для подсчета (нужно было оценивать входные данные, выходные данные, файлы и т.п.). Впрочем, на пути широкого использования функциональных точек встали не трудозатраты, а их сложность. По моему мнению, именно увеличение сложности подсчета — беглый просмотр веб-сайта International Function Point User Group (IFPUG) дает хорошее представление об уровне этой сложности — привело к сокращению использования этого показателя.

Так или иначе, потребность в оценке «размера» программного проекта никуда не исчезла. Проблема с обоими историческими показателями имеет две стороны — их сложно определять и они основаны на каскадном подходе к разработке. Нам же требуется показатель размера, который просто определить и можно применять без необходимости проходить все требования и фазы проектирования.

Два критически важных отличия пунктов от количества строк программы и функциональных точек заключаются в том, что они проще в расчете и могут определяться на значительно более раннем этапе. Почему они проще? Да потому, что характеризуют относительный размер, а не

абсолютный. Почему их можно определять на более раннем этапе? Да потому, что они относятся в большей мере к относительному размеру, чем к абсолютному. Как подчеркивает Майк, оценка с использованием пунктов — это обсуждение историй за круглым столом (обмен знаниями) и предположительная оценка относительного размера истории. Оценка относительного размера, в отличие от оценки абсолютного, осуществляется на удивление быстро. К тому же после нескольких итераций оценки размера и поставки продукта точность предположительных оценок, даваемых командой, значительно возрастает. Описание Майком «как» и «почему» при сравнении оценки в пунктах и идеальных днях позволяет глубоко понять эту критически важную тему.

Еще одним примером тщательности изложения материалов Майком служат главы 9–11, посвященные приоритизации историй. Майк не ограничивается советом браться в первую очередь за истории с наивысшей стоимостью, а раскрывает ключевые аспекты стоимости: финансовые выгоды, затраты, инновации/знание и риск. Он дает четкие разъяснения по каждому из этих аспектов (включая общие представления о чистой приведенной стоимости, внутренней ставке доходности и других инструментах финансового анализа), а затем приводит ряд схем (с разной степенью упрощения) принятия решений по весам на основе рассмотренных аспектов стоимости.

Нередко новички в области agile-разработки полагают, что если применить определенную методологию 12, 19 или 8 раз, то это и будет Agile, Extreme, Cristal Clear или еще что-нибудь подобное. Однако на самом деле вы применяете методологию Agile, Extreme и т.п. только тогда, когда знаете достаточно, чтобы адаптировать ее к своей конкретной ситуации. Суть agile-разработки — непрерывное обучение и адаптация. Что Майк отлично делает в этой книге, так это знакомит нас с идеями и практикой, которые помогают вывести наши agile-оценку и agile-планирование на следующий уровень развития. Майк детально разъясняет, «как» подойти к делу, например провести оценку в пунктах и идеальных днях, и «почему» нужно подходить именно так, представляя плюсы и минусы показателей «пункты» и «идеальные дни». Хотя он обычно дает рекомендации (сам он предпочитает пункты), в книге достаточно информации, чтобы мы уверенно могли адаптировать методику к своей ситуации.

Именно в этом и заключается вклад Майка в данную область — сначала он помогает нам получать новые знания и опыт через оценку и планирование («как»), а затем принимать решения об использовании нового знания для адаптирования оценок и планов к новым, уникальным или просто конкретным ситуациям («почему»). Из полудесятка книг, которые я постоянно рекомендую своим клиентам, две принадлежат Майку. «Agile-подход к оценке и планированию» входит в мой список обязательных

для изучения материалов для тех, кто хочет понять последние веяния в сфере agile-управления проектами.

Джим Хайсмит,
директор по agile-практике в компании Cutter Consortium,
Флагстафф, Аризона, август 2005 г.

Предисловие

«Agile-подход к разработке не годится для Yahoo!, за исключением, быть может, небольших оперативных проектов, поскольку команды совершенно не занимаются планированием, а члены команд не могут оценить проделанную работу. Им приходится давать указания, что делать».

Это реальное высказывание, которое я неоднократно слышал, когда начинал руководить внедрением agile-подхода в Yahoo!. Люди, которые не понимают концепцию agile-разработки, полагают, что это просто отказ от документации и планирования, лицензия на свободное плавание. На самом деле такое представление предельно далеко от истины.

«Команды совершенно не занимаются планированием». Тот, кто говорит такое, забывает, что agile-команда раз в две недели посвящает полдня составлению перечня задач, которые необходимо выполнить, чтобы представить определенную полезную для пользователя функциональность через две недели. То, что команды распределяют процесс планирования на весь период реализации проекта, воспринимается как отсутствие планирования. Это не так, и agile-команды в Yahoo! создают продукты, которые нравятся нашим менеджерам по продуктам намного больше, чем продукты традиционных команд.

«Члены команд не могут оценить проделанную работу. Им приходится давать указания, что делать». Это классическое заблуждение. Упование на магическую способность менеджера по продукту или руководителя проекта предвидеть, что кто-то другой, эксперт в своем деле, может реально создать, самоубийственно для заказчика. Нередко такой подход на деле оборачивается простым соглашательством с нереалистичными целями заказчика. Члены команды при этом вынуждены работать круглые сутки и халтурить. А потом мы удивляемся, почему в нашей отрасли люди так быстро выгорают и почему так низок моральный дух.

Оценка и планирование — темы, по которым мне задают больше всего вопросов, особенно во вновь созданных командах. Простой подход к планированию не только итерации, но и проекта в целом — вещь, прямо скажем, неопределимая. Менеджеров по продукту волнует достижение целевых показателей по выручке и предсказуемое планирование релизов. Команды могут использовать гибкий подход и изменять направление разработки при необходимости, однако очень важно иметь стратегический

план, которому они будут следовать. Кому нужно быстрое движение, если вы идете не в том направлении? Освоение искусства оценки сделанного и планирования дальнейших действий — важнейшее условие успеха для тех, кто хочет внедрить agile-подход в своей организации.

Учебный курс Майка по оценке и планированию пользуется наибольшей популярностью при освоении agile-подхода в Yahoo!. Он дает командам профессиональные знания и инструменты, необходимые, чтобы заниматься планированием именно в таком объеме, который требуется для оптимизации результатов. Это и в самом деле работает, если следовать рекомендациям Майка? Да. Эффект от применения agile-подхода в Yahoo! грандиозен. Команды, прошедшие курс Майка, сразу начинают использовать его рекомендации на практике. Мы стали выводить продукты на рынок быстрее, а команды буквально в восторге от agile-подхода.

Почему agile-подход к оценке и планированию более эффективен, чем традиционные методы? Он сконцентрирован на создании стоимости и формировании доверительных отношений между заказчиком и проектными командами. Обеспечение полной прозрачности и информирование об изменениях по мере их появления позволяют заказчику принимать наилучшие решения. На предыдущем месте работы я воочию наблюдал процесс перехода из вечного хаоса в состояние предсказуемости, когда мы действительно стали браться за проекты, которые могли выполнить. Заказчикам не каждый раз нравились наши ответы (им ведь всегда хочется получить продукт уже завтра), однако они как минимум верили, что мы говорим правду, и не чувствовали, что их постоянно водят за нос.

Эта книга предельно честна. В ней не говорится о том, как достичь абсолютной точности оценок. Это была бы напрасная трата сил в стремлении достичь недостижимого. Майк не пытается навязать вам удобные шаблоны, в которых остается только заполнить пробелы. Вместо этого он заставляет вас думать и учиться тому, как подходить к проблемам и успешно решать их. Одинаковых проектов и организаций не бывает, поэтому намного важнее научиться мыслить определенным образом и применять принципы. Майк перенес свой огромный опыт, полученный из реальной практики, и индивидуальный подход на страницы этой книги. Она живая и честная. Она определенно должна стоять в первых строках вашего списка учебной литературы.

Габриэль Бенефилд,
директор Agile Product Development Yahoo!

Благодарности

Я выражаю глубокую признательность официальным рецензентам этой книги. Том Поппендик, Стив Токи, Пол Ходжеттс, Майк Сирфос и Престон Смит дали мне полезные отзывы и рекомендации. Благодаря их вкладу книга стала намного лучше. В частности, я хочу поблагодарить Стива и Тома за то, что они не ограничились формальными обязанностями. Стив отметил целый ряд идей и концепций, которые я упустил, и подсказал несколько полезных источников информации. Но главное, он натолкнул меня на то, что стало впоследствии моей мантрой при проведении занятий по оценке и планированию: оценивай размер, определяй срок. Том, пожалуй, потратил на эту книгу больше сил, чем я. Он неустанно подчеркивал важность того, чтобы книга была ориентирована на команду в целом, а не только на руководителя проекта. Именно в разговорах с Томом я понял, что книга по планированию должна быть шире, чем просто ответ на вопрос «Когда мы уже закончим?». В широком контексте создания стоимости для наших организаций дать ответ на этот вопрос нетрудно.

Мой поклон Джону Гудсену из RADSoft. Первоначально мы с Джоном собирались писать эту книгу вместе. Увы, наши календарные графики не позволили сделать этого, однако я благодарен Джону за обсуждения набросков книги.

Одно из величайших благ интернета — возможность показывать книгу другим в процессе работы над ней. Рукопись этой книги висела на моем веб-сайте в течение 20 месяцев, и комментарии и замечания читателей здорово помогли довести ее до ума. Я особенно благодарен Брайану Амброджиано, Кену Ауэру, Саймону Бейкеру, Рэю Боэму, Лесли Боррелл, Кларку Чингу, Лайзе Криспин, Рейчел Дейвис, Майку Дуайеру, Хакану Эрдогмусу, Джону Фаваро, Крису Гарднеру, Джону Джилману, Свену Гортсу, Полу Грю, Сридхару Джайяраману, Анджело Кастроулису, Лайзе Катценмейер, Лассе Коскеле, Митчу Лейси, Патрику Логану, Кенту Макдональду, Эрику Петерсену, Майку Поулену, Дж. Рейзенбергу, Биллу Рамосу, Мэтту Риду, Джорджу Рейлли, Крису Риммеру, Оуэну Роджерсу, Кевину Резерфорду, Дику Шилу, Джеймсу Шилу, Кену Скотту, Карлу Скотланду, Алану Шаллоуэю, Джагадишу Шринивасавадхани, Мишель Слайджер, Карен Смайли, Хьюберту Смитсу, Виктору Сзалвею, Чарли Трейнору, Раджу Вагхрею, Рудигеру Вульффу, Скотту Уорли и Джейсону

Йипу.

Я также хотел бы поблагодарить всех, кто принимал участие в проведении моего учебного курса по agile-подходу к оценке и планированию в последние два года, как в компаниях, так и на конференциях. Моя благодарность адресована также всем клиентам, особенно тем, у которых я проводил занятия по оценке и планированию и которые пользуются идеями из этой книги, в том числе таким компаниям, как Farm Credit Systems of America, Fast401k, High Moon Studios, Nielsen Media Research, Sun Microsystems, Ultimate Software, Vision Pace, Yahoo! и Webroot.

Как всегда, с командой издательства Prentice Hall было очень приятно работать. Пол Петралиа и Мишель Хаусли участвовали в проекте от начала до конца. Тиррелл Албо помогла преодолеть некоторые трудности с использованием системы FrameMaker. Я попросил прикрепить ко мне придирчивого редактора, чтобы книга получилась предельно хорошей. Этим редактором оказалась Кэти Симпсон, которая сделала именно то, что мне хотелось. Наконец, Лара Уайсон тщательно контролировала весь процесс превращения рукописи в книгу, которую вы держите в руках. Она без устали отвечала на мои бесконечные вопросы и электронные письма.

Благодарю Боба Мартина за включение этой книги в свою серию. Дядя Боб — один из моих любимых писателей еще с той поры, когда он был редактором журнала *C++ Report*. Боб сделал для распространения идей agile-подхода в сообществе разработчиков программного обеспечения столько, что попасть в его серию книг — большая честь. Также я хочу поблагодарить Джима Хайсмита из Cutter Consortium и Габриэля Бенефилда из Yahoo! за предисловия. Работа с ними была удовольствием.

Что бы я ни сказал, этого будет мало, чтобы выразить признательность моим близким, создавшим условия для работы над этой книгой. Предполагалось, что значительную часть подготовки рукописи я сделаю во время своих поездок. Однако все сложилось иначе. Лоре, моей жене и партнеру во всех начинаниях, пришлось не покладая рук заниматься моими делами, делами нашей компании и этой книгой. Она многократно читала и перечитывала каждую главу. Без ее помощи я бы не сделал и десятой части того, что мы сделали вместе. Мои чудесные дочери Саванна и Делани настолько привыкли к постоянному уединению своего отца в кабинете, что мое появление стало казаться им странным. Я благодарю их за нежные объятия и поцелуи и за то, что они такие, какие есть.

Введение

Эта книга в основном посвящена планированию, под которым я понимаю ответ на вопрос «Что необходимо сделать и к какому сроку?». Однако ответ на него требует ответа на вопросы, связанные с оценкой («Насколько это объемно?») и составлением календарного графика («Когда это должно быть сделано?» и «Сколько будет готово к этому моменту?»).

Книга состоит из семи частей, в ней 23 главы. Каждая глава завершается обобщением ключевых моментов и вопросами для обсуждения. Поскольку оценка и планирование — это виды деятельности, которыми занимается команда в целом, я предполагаю, что книгу будут читать все ее члены, а потом собираться (возможно, раз в неделю) для обсуждения прочитанного и вопросов в конце каждой главы. Учитывая, что agile-подход к разработке программного обеспечения популярен во всем мире, я старался избежать чрезмерной концентрации внимания на США. С этой целью в книге используется универсальное обозначение валюты и денежные суммы выглядят как \$500, а не \$500, €500 и т.п.

В части I рассказывается, почему планирование так важно, рассматриваются проблемы, которые часто встречаются на практике, и цели agile-подхода. В главе 1 говорится о цели планирования, о том, что нужно для составления хорошего плана и что превращает планирование в гибкий процесс. Основные причины, по которым традиционные подходы к оценке и планированию дают неудовлетворительные результаты, рассматриваются в главе 2. Наконец, глава 3 содержит краткий перечень особенностей гибких методов и описание обобщенных принципов agile-подхода к оценке и планированию, которому посвящаются последующие части книги.

В части II представлено основное правило оценки, требующее, чтобы оценки размера и срока никогда не смешивались. Главы 4 и 5 знакомят читателя с пунктами и идеальными днями — двумя показателями, подходящими для оценки размера разрабатываемых компонентов. В главе 6 описываются методы оценки в пунктах и идеальных днях и дается представление о покере планирования. Глава 7 посвящена тому, когда и как проводить переоценку, а в главе 8 приводятся рекомендации, что лучше выбрать — пункты или идеальные дни.

Часть III «Планирование на основе стоимости» содержит рекомендации

для проектной команды относительно того, как создать наилучший конечный продукт. В главе 9 перечислены факторы, которые необходимо учитывать в процессе приоритизации функций. В главе 10 представлен подход к моделированию финансовой отдачи от отдельной функции или набора функций, а также методы сравнения финансовой отдачи, с тем чтобы в первую очередь разрабатывались наиболее ценные функции. В главу 11 включены рекомендации по оценке и последующей приоритизации желательности функцией для пользователей продукта. Глава 12 завершает раздел обсуждением вопроса о том, как разбивать крупные функции на более мелкие части, которыми легче управлять.

В части IV мы переключаем внимание на вопросы, связанные с составлением календарных графиков для проекта. Глава 13 открывается обзором этапов составления календарных графиков для сравнительно простого, выполняемого одной командой проекта. В следующей главе (14) рассматривается вопрос планирования итераций. Главы 15 и 16 посвящены выбору длины итераций для проекта и оценке первоначального темпа продвижения команды. В главе 17 детально разбирается составление календарных графиков для проекта с высоким уровнем неопределенности или с высокой чувствительностью к некорректности графика. Раздел завершается главой 18, в которой описываются дополнительные этапы при оценке и планировании проекта, осуществляемого несколькими командами.

После составления плана необходимо проинформировать о нем всю организацию и использовать его для мониторинга прогресса разработки. Именно этим вопросам посвящены три главы части V. В главе 19 рассматривается мониторинг плана релиза, а в главе 20 — плана итераций. В последней главе этого раздела (21) разбираются вопросы информирования о плане и процессе его выполнения.

Часть VI состоит всего из одной главы — 22, в которой рассматривается вопрос, почему работает agile-подход к оценке и планированию. Эта глава является своего рода дополнением к главе 2, где говорится о том, почему традиционные подходы нередко дают неудовлетворительные результаты.

Заключительная часть (VII) также включает в себя только одну главу. Глава 23 представляет собой расширенный анализ примера, который повторяет основные моменты этой книги применительно к гипотетической ситуации.

Часть I

Проблема и цель

Чтобы уяснить суть agile-подхода к оценке и планированию, необходимо ясно понимать цель планирования. Именно этому вопросу посвящена глава 1 данного раздела. В главе 2 рассматриваются основные причины, по которым традиционные подходы к оценке и планированию дают неудовлетворительные результаты. Заключительная глава этого раздела содержит описание обобщенных принципов agile-подхода к оценке и планированию, которому посвящаются последующие части книги.

Глава 1

Цель планирования

Планирование — это все. Планы — ничто.

Фельдмаршал Хельмут фон Мольтке

Оценка и планирование критически важны для успеха проекта по разработке программного обеспечения любого размера и значимости. Планы определяют наши инвестиционные решения: мы можем взяться за проект, на выполнение которого, по нашим оценкам, потребуется полгода и \$1 млн [1], и отказаться от этого же проекта, если на него потребуется два года и \$4 млн. Планы помогают нам понять, кого нужно привлечь к работам по проекту в течение определенного периода. Планы помогают нам понять, как продвигается создание функциональности, которая нужна пользователям и получения которой они ожидают. Без планов мы открываем ворота для целого ряда проблем.

Процесс планирования, однако, сложен, а планы нередко получаются далекими от реальности. Как результат, команды зачастую впадают в одну из двух крайностей: они либо полностью отказываются от планирования, либо тратят столько сил на составление планов, что начинают верить в их правильность. Команды, которые отказываются от планирования, не могут ответить на такие фундаментальные вопросы, как «Когда это должно быть выполнено?» и «Можем ли мы ожидать выпуск продукта в июне?». Команда, затратившая слишком много сил на планирование, обольщает себя уверенностью в том, что план в принципе может быть «правильным». Ее план может быть тщательно проработанным, но вовсе не обязательно отличаться высокой точностью или полезностью.

Тот факт, что оценка и планирование — дело непростое, не является открытием. Это известно давным-давно. В 1981 г. Барри Боэм построил первую версию того, что Стив Макконнелл (Steve McConnell, 1998) позднее назвал «конус неопределенности». На рис. 1.1 показаны первоначальные диапазоны неопределенности Боэма в разных точках в процессе последовательного развития («каскадный процесс»). Конус неопределенности говорит о том, что на этапе оценки осуществимости проекта оценка обычно отклоняется от истины на 60–160%. Иначе говоря,

на проект, который, как ожидается, должен занять 20 недель, может потребоваться от 12 до 32 недель. После формулирования требований в письменном виде оценка может отклоняться на $\pm 15\%$ в любом направлении, т.е. плановый срок 20 недель может сократиться до 17 недель или вырасти до 23 недель.

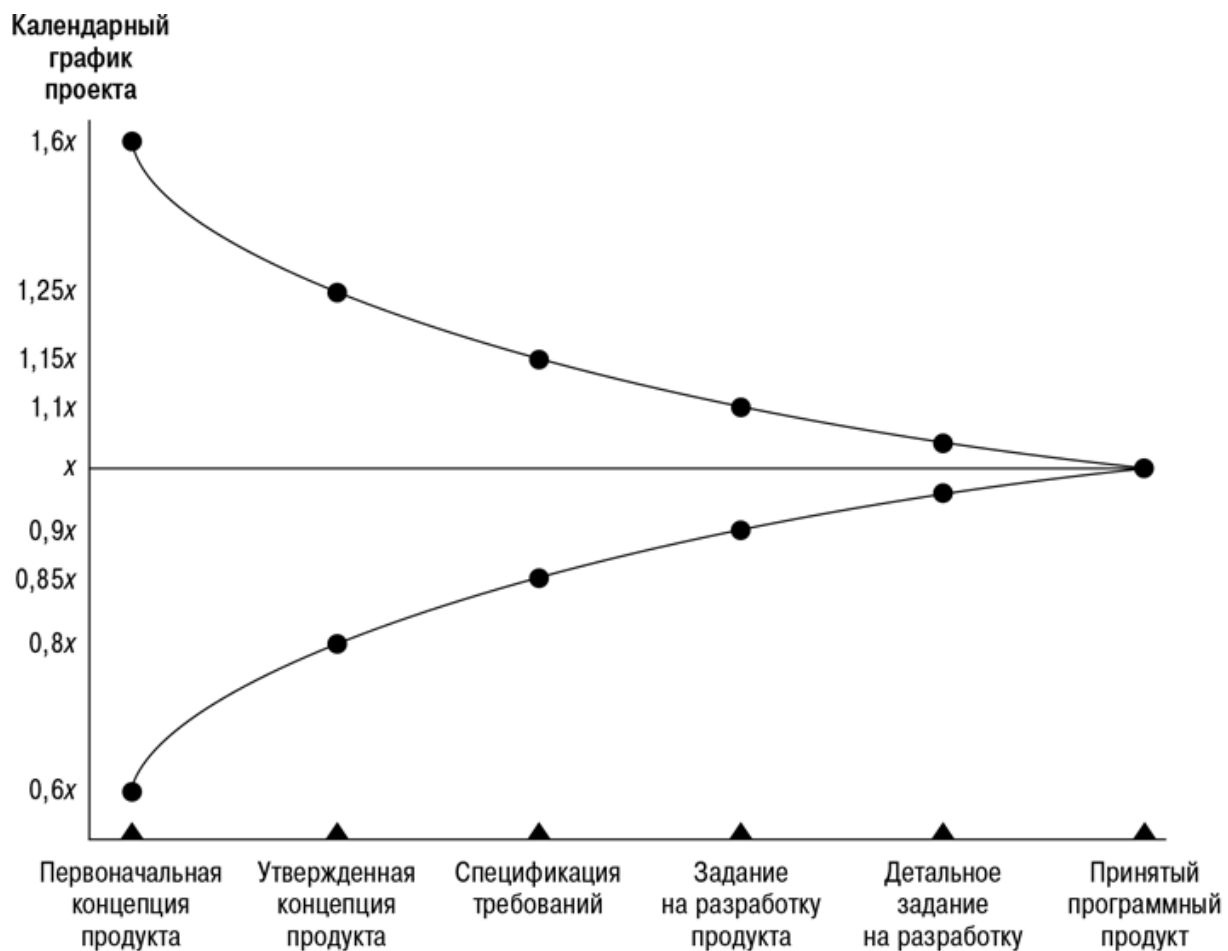


Рис. 1.1. Конус неопределенности сужается по мере выполнения проекта

Институт управления проектами (Project Management Institute — PMI) имеет сходную точку зрения на постепенное повышение точности оценок, однако он считает, что конус неопределенности должен быть асимметричным. PMI предлагает принимать начальный уровень отклонений оценки в диапазоне от $+75\%$ до -25% . Следующий этап — бюджетные предположения — предполагает диапазон отклонений от $+25\%$ до -10% , за ним следует этап окончательной бюджетной оценки с диапазоном отклонений от $+10\%$ до -5% .

Зачем это нужно

Если оценка и планирование настолько трудны и если точную оценку невозможно получить вплоть до последней фазы выполнения проекта, то зачем этим заниматься? Конечно, очевидной причиной является то, что в организациях, где мы работаем, от нас нередко требуют предоставления оценок. Планы и графики могут требоваться для таких вполне понятных целей, как планирование маркетинговых кампаний, планирование релизов продукта и обучение внутренних пользователей. Это очень важные потребности, и трудность оценки проекта не может служить основанием для отказа от составления плана или графика, который организация может использовать для их удовлетворения. Вместе с тем помимо этих номинальных потребностей существует значительно более фундаментальная причина не жалеть сил на оценку и планирование.

Оценка и планирование — это не просто определение сроков или календарных графиков. Планирование, особенно непрерывное планирование итераций, — это поиск стоимости. Планирование представляет собой попытку найти оптимальное решение всеобъемлющего вопроса разработки продукта: что мы должны создать? Для ответа на этот вопрос команда анализирует функциональность, ресурсы и сроки. Ответ на данный вопрос нельзя найти одномоментно. Его ищут итерационно, шаг за шагом. В начале проекта мы, например, можем решить, что продукт должен иметь определенный набор функций, а его выпуск должен состояться 31 августа. Однако в июне оказывается, что лучше выпустить продукт немного позднее, но с более полным набором функций. А может наоборот: лучше сократить набор функций, но выпустить продукт чуть раньше.

Хороший процесс планирования поддерживает такой подход, обеспечивая:

- сокращение риска;
- снижение неопределенности;
- создание условий для принятия более качественных решений;
- формирование доверия;
- распространение информации.

Сокращение риска

Планирование повышает вероятность успеха проекта, обеспечивая идентификацию проектных рисков. Одни проекты настолько рискованны, что лучше не браться за них. Другие могут содержать функциональности, риски которых, если к ним подойти должным образом с самого начала, поддаются ограничению.

На обсуждениях, происходящих в процессе оценки, поднимаются вопросы, которые позволяют выявить подводные камни проекта. Допустим,

вас просят оценить, сколько времени потребуется на интеграцию нового проекта со старой, построенной на основе мейнфрейма системой, о которой вы ничего не знаете. Это заставляет смотреть на функцию интеграции как на потенциальный риск. Проектная команда может устранить этот риск сразу, потратив определенное время на знакомство со старой системой. Альтернативно риск можно идентифицировать и учесть его в работе как отдельную величину или в виде диапазона и включить в общую неопределенность и риск.

Снижение неопределенности

В процессе реализации проекта команда создает новые функциональные возможности продукта, а также генерирует новые знания о продукте, используемых технологиях и своих собственных квалификациях. Крайне важно идентифицировать эти знания и учитывать их при итеративном планировании, которое должно помогать команде улучшать ее представления о продукте. Самым серьезным риском большинства проектов является риск создания несоответствующего продукта. Этот риск, однако, чаще всего полностью игнорируется. Agile-подход к планированию позволяет кардинально уменьшить (а в идеале устранить) такой риск.

Нередко цитируемые исследования Chaos Report (Standish Group, 2001) определяют успешный проект как такой, который выполнен в срок и в рамках бюджета и имеет все изначально предусмотренные функциональности. Это опасное определение, поскольку оно не учитывает того, что функциональность, казавшаяся хорошей до начала проекта, может оказаться не стоящей вложений, когда команда реально возьмется за дело. Если бы меня попросили дать определение неудачному проекту, то в числе прочего я бы назвал «проект, в котором никто не высказал более удачных идей, чем включенные в исходный перечень требований». Мы приветствуем такие проекты, в которых инвестиции, календарные графики и решения по функциональностям периодически переоцениваются. Проект, имеющий все предусмотренные в первоначальном плане функциональности, не обязательно успешен. Пользователи продукта и клиент вряд ли будут довольны, если хорошие новые функциональности будут принесены в жертву средненьким просто потому, что те заложены в первоначальный план.

Условия для принятия более качественных решений

Оценки и планы помогают нам принимать решения. Как организации определить, стоит ли браться за тот или иной проект, не имея оценки стоимости и затрат по проекту? Помимо поддержки решений относительно

принятия проектов оценки позволяют гарантировать, что мы работаем над самыми ценными проектами. Предположим, что организация рассматривает два проекта, один из которых оценивается в \$1 млн, а другой в \$2 млн. Во-первых, календарные графики и оценки затрат нужны организации для того, чтобы определить, есть ли смысл заниматься этими проектами; не окажутся ли они настолько продолжительными, что не уложатся в рыночное окно? Не окажутся ли они настолько затратными, что потеряют смысл? Во-вторых, оценки и план нужны организации для того, чтобы решить, за какой проект взяться. Может оказаться, что у организации есть возможности реализовать один проект, оба проекта или ни один из них, если затраты слишком высоки.

Оценки необходимы организации для принятия и других решений, помимо решения о том, браться за проект или нет. В некоторых случаях штат исполнителей проекта более важен, чем календарный график. Так, реализация проекта может потерять смысл, если она требует участия ведущего разработчика организации, который полностью занят в другом проекте. Вместе с тем если удастся составить план, показывающий, как обойтись без участия ведущего разработчика, то за реализацию проекта, возможно, стоит взяться.

Многие решения, принимаемые в процессе планирования проекта, являются компромиссными. Например, в любом проекте неизбежно приходится искать компромисс между временем разработки и затратами. Нередко наименее затратный путь разработки системы — это нанять одного хорошего программиста и позволить ему работать над созданием продукта 10 или 20 лет с возможностью отвлекаться на освоение соответствующей профессиональной сферы, совершенствование в области администрирования баз данных и т.п. Очевидно, однако, что возможность ждать появления готового продукта 20 лет редко когда выпадает, поэтому мы поручаем работу команде. Команде из 30 человек, возможно, потребуется год (30 человеко-лет) на разработку, с которой один программист мог бы справиться за 20 лет. Стоимость разработки при этом возрастает, однако стоимость, создаваемая при получении продукта на 19 лет раньше, покрывает увеличение затрат.

Нам постоянно приходится принимать компромиссные решения в отношении функциональности, трудозатрат, издержек и времени. Стоит ли из-за той или иной функции откладывать выпуск версии продукта? Следует ли нам привлечь к проекту еще одного разработчика, чтобы включить конкретную функцию в ближайший релиз? Следует ли выпустить версию в июне или стоит отложить выпуск до августа и включить в нее дополнительную функцию? Следует ли нам купить данное средство разработки? Для принятия решений нам необходимы оценки как затрат, так и выгод.

Доверие

Частая надежная поставка обещанной функциональности рождает доверие между разработчиками и заказчиками продукта. Достоверные оценки обеспечивают надежность поставки. Оценки нужны клиенту для распределения приоритетов и принятия компромиссных решений. Оценки также помогают клиенту решить, сколько функций разрабатывать. Вместо того, чтобы потратить 20 дней и получить все, может быть, лучше ограничиться 10 днями и получить 80% выгод. Клиенты с неохотой идут на принятие подобных компромиссных решений на начальной стадии осуществления проекта, если оценки разработчиков не внушают доверия.

Достоверные оценки позволяют разработчикам двигаться в стабильном темпе. Результатом является высококачественная программа и снижение количества ошибок. Это, в свою очередь, повышает достоверность оценок, поскольку на такую во многом непредсказуемую работу, как устранение ошибок, приходится тратить меньше времени.

Распространение информации

План дает представление об ожиданиях и показывает, что может произойти в процессе выполнения проекта. План не гарантирует получения точного набора функций в точно определенную дату по заданной стоимости. Он содержит информацию и устанавливает набор базовых ожиданий. Планы, к сожалению, зачастую сводятся к определению конкретной даты, а все допущения и ожидания, которые привели к появлению на свет этой даты, забываются.

Допустим, вы спрашиваете меня, когда будет завершен проект. Я говорю вам, что через семь месяцев, однако не объясняю, каким образом у меня получился именно такой срок. Вы, конечно, скептически отнесетесь к моей оценке. Без дополнительной информации вам не удастся определить, хорошо ли я продумал этот вопрос и реалистична ли моя оценка.

Теперь представьте, что я представляю вам план, который оценивает срок завершения работ в семь-девять месяцев, показывает, какая работа будет выполнена в первые один или два месяца, содержит перечень ключевых допущений и обрисовывает наш совместный подход к оценке прогресса. В этом случае вы можете проанализировать мой план и понять, насколько ему можно доверять.

Что делает план хорошим

Хорошим считается такой план, который, по мнению заинтересованных

сторон, является достаточно надежным для того, чтобы на его основе принимать решения. На начальном этапе осуществления проекта это может быть указание на то, что продукт будет выпущен скорее в третьем квартале, а не во втором и что он будет иметь примерно обрисованный набор функций. На более позднем этапе работ этот план, чтобы оставаться по-прежнему полезным для принятия решений, должен быть более точным.

Предположим, вы оцениваете и планируете новый релиз флагманского продукта компании. По вашим расчетам, новая версия будет готова к выпуску через шесть месяцев. Вы составляете план с описанием набора функций, которые определено будут иметься в новой версии продукта, и еще одного набора функций, которые могут быть включены в продукт в зависимости от успешности процесса разработки.

Другие сотрудники компании могут использовать этот план для принятия решений. Они могут готовить маркетинговые материалы, планировать рекламную кампанию, выделять ресурсы на переобучение ключевых клиентов и т.п. Этот план полезен до тех пор, пока он реально предсказывает, что будет происходить в процессе работы над проектом. Если разработка займет 12 месяцев вместо запланированных шести, то план нельзя назвать хорошим.

Вместе с тем если проект займет семь месяцев вместо шести, то план в определенной мере полезен. Да, он неточен и мог привести к принятию не совсем правильных решений. Однако поставка продукта через семь месяцев при осуществлении проекта с расчетным сроком шесть месяцев вовсе не конец света и определенно укладывается в пределы допустимой погрешности PMI для бюджетных оценок. План, несмотря на его неточность, был бы еще полезнее, если бы он регулярно корректировался по мере выполнения проекта. В этом случае задержка поставки на один месяц не стала бы неожиданностью ни для кого.

Что делает планирование гибким

Эта книга посвящена agile-подходу к планированию, а не гибким планам. Планы — это документы и цифры, статическое описание наших представлений о развитии проекта в неопределенном будущем. Планирование — это вид деятельности. Agile-подход предполагает перенос акцента с планов на процесс планирования.

Agile-подход позволяет сбалансировать вкладываемые в планирование усилия и трудозатраты с учетом того, что план будет пересматриваться в процессе осуществления проекта. Никто не собирается менять план ради изменений, изменения вносятся потому, что мы получаем новую

информацию или исправляем ошибки. Мы можем получить информацию, например, о том, что пользователи хотят расширить конкретную функцию или, наоборот, урезать ее, или узнать, что простота использования продукта значительно важнее, чем казалось вначале, или обнаружить, что программирование на новом языке занимает больше времени, чем ожидалось. Финансовые последствия каждого такого изменения можно оценить и, если это целесообразно, изменить план и календарный график.

Вновь выясненные обстоятельства влияют на наши планы. Это означает, что нам нужны такие планы, которые можно легко изменять. Именно поэтому процесс планирования становится более важным, чем сам план. Знания и представления, которые мы получаем в процессе планирования, продолжают существовать и после того, как от старого плана отказываются и заменяют его новым. Таким образом, под гибким понимают такой план, который легко поддается изменениям.

Изменение плана само по себе не означает изменение дат. Мы можем сделать это, а можем и не делать. Однако, если оказывается, что мы заблуждались в отношении определенного аспекта целевого продукта и нужно устранить ошибку, в план необходимо внести изменения. Существуют разные способы корректировки плана без изменения даты. Можно отказаться от функции, можно сократить ее объем, можно увеличить численность работников, занятых в проекте, и т.д.

Поскольку мы признаем, что не можем с абсолютной точностью определить все аспекты проекта с самого начала, нам не нужно пытаться запланировать все и вся на начальном этапе. При agile-подходе планирование осуществляется более или менее равномерно на протяжении всего срока реализации проекта. Вслед за планированием релиза, закладывающим фундамент, выполняется серия раундов планирования, и весь процесс многократно повторяется по мере осуществления проекта.

Итак, при определении agile-подхода к планированию мы установили, что он:

- фокусируется на планировании, а не на плане;
- поощряет изменения;
- приводит к составлению планов, легко поддающихся изменению;
- распределяет процесс планирования по всему сроку осуществление проекта.

Резюме

Оценка и планирование критически важны, однако сложны и подвержены ошибкам. Так или иначе, отказываться от них просто из-за трудности

нельзя. Оценки, данные в начале проекта, значительно менее точны, чем оценки, полученные позднее. Графическое представление процесса постепенного повышения точности оценок называют *конусом неопределенности*.

Цель планирования — получение оптимального ответа на глобальный вопрос разработки продукта — вопрос о том, что именно создавать. Ответ включает в себя описание функций, ресурсов, а также календарный график. Ответ на этот вопрос, подкрепленный снижающим риск процессом планирования, сокращает неопределенность, дает основу для объективного принятия решений, устанавливает доверие и обеспечивает распространение информации.

Хорошим является такой план, который достаточно надежен для того, чтобы на его основе принимать решения относительно продукта и проекта. Agile-подход к планированию сфокусирован больше на планировании, а не на создании плана, поощряет изменения, приводит к составлению планов, легко поддающихся изменению, и распределяет процесс планирования по всему сроку осуществления проекта.

Вопросы для обсуждения

1. Эта глава открывается утверждением о том, что чрезмерное планирование и отсутствие планирования одинаково опасны. Какой объем планирования оптимален для вашего текущего проекта?
2. Какие еще доводы в пользу планирования вы можете привести?
3. Вспомните один или два наиболее успешных проекта, в которых вы участвовали. Какую роль планирование играло в этих проектах?

Глава 2

Почему планирование дает неудовлетворительные результаты

Ни один план не выдерживает реального столкновения с противником.

Фельдмаршал Хельмут фон Мольтке

В предыдущей главе говорилось, что целью планирования является итеративное приближение к получению оптимального ответа на вопрос о разработке совершенно нового продукта — вопрос о том, что именно создавать. Иначе говоря, какими функциональными возможностями должен обладать продукт, за какой срок его необходимо создать и сколько для этого потребуется ресурсов. Мы узнали, что планирование поддерживает этот процесс, обеспечивая снижение риска, уменьшение неопределенности в отношении облика продукта, создание основы для принятия более качественных решений, укрепление доверия и распространение информации.

К сожалению, традиционные подходы к планированию нередко подводят нас. Отвечая на комплексный вопрос об объеме / календарном графике / ресурсах по новому продукту, наши традиционные процессы планирования не всегда дают удовлетворительные ответы и продукты. В подтверждение сказанного приведу следующие данные:

- почти две трети проектов значительно превышают сметы затрат (Lederer and Prasad, 1992);
- 64% функций, включенных в продукты, используются редко или вообще не используются (Johnson, 2002);
- срок выполнения среднего проекта превышает календарный график на 100% (Standish, 2001).

В этой главе мы рассмотрим пять причин, по которым планирование дает неудовлетворительные результаты.

Планирование ориентировано на деятельность, а не на функцию

Критическая проблема традиционных подходов к планированию заключается в том, что они сфокусированы на выполнении той или иной деятельности, а не на поставке функциональности. Диаграмма Гантта для традиционно управляемого проекта, или структура распределения работ, идентифицирует виды деятельности, подлежащие выполнению. Именно по ней мы определяем прогресс команды. Первая проблема планирования по видам деятельности связана с тем, что клиенты не получают никакой стоимости от выполнения видов деятельности. Единицей стоимости для клиента является функция. Планирование, таким образом, должно осуществляться на уровне функций, а не видов деятельности.

Вторая проблема возникает при анализе ранее составленного традиционного календарного графика. Когда мы анализируем календарный график, в котором представлены виды деятельности, наше внимание приковано к поиску пропущенных видов деятельности, а не отсутствующих функций.

Дополнительные проблемы объясняются тем, что планы на основе видов деятельности (процессно-ориентированные планы) зачастую ведут к проектам, которые не укладываются в календарные графики. Сталкиваясь с невозможностью выдержать сроки, заложенные в календарный график, некоторые команды пытаются сэкономить время за счет неуместного снижения качества. Существует также практика принятия политики, которая ограничивает возможности внесения изменений в продукт, в том числе и очень ценных изменений. Вот основные причины, по которым в результате планирования по видам деятельности трудно уложиться в сроки, предусмотренные календарным графиком:

- запланированные работы не завершаются досрочно;
- запаздывание распространяется на последующие этапы календарного графика;
- работы не являются независимыми.

Каждая из этих проблем рассматривается в последующих разделах.

Запланированные работы не завершаются досрочно

Несколько лет назад я занимался двумя крупными проектами, которые отнимали много времени. Мне нужно было запрограммировать ряд интересных функций для продукта, а также подготовить документацию для аудита соответствия требованиям стандарта ISO 9001. Если

программирование доставляло мне удовольствие, то подготовка документов — нет. Неудивительно, что я умудрился раздуть объем программирования так, что оно заняло чуть ли не все мое время и практически вытеснило подготовку к аудиту.

Я вовсе не одинок в таком подходе к работе. Если говорить начистоту, то подобное поведение настолько распространено, что у него есть даже свое название — закон Паркинсона (1993 г.). Этот закон гласит:

«Работа растягивается так, чтобы занять все отведенное на нее время».

Паркинсон говорит, что нам требуется столько времени на завершение какого-либо дела, сколько, на наш взгляд, будет позволено. Если на стене висит диаграмма Ганта, из которой следует, что на тот или иной вид деятельности отведено пять дней, то программист, которому поручена эта работа, будет стараться растянуть удовольствие на полные пять дней. Чтобы избежать досрочного завершения, он может, например, добавить в программу какие-нибудь лишние функции (практика, известная как *украшательство*). Или может использовать часть времени на изучение какой-нибудь новой технологии, которая, на его взгляд, полезна для дела. Единственное, на что он редко когда идет, так это досрочное завершение работы. Во многих организациях в случае досрочного завершения работы шеф может обвинить исполнителя в предоставлении раздутой оценки. Или, как вариант, шеф станет рассчитывать на досрочное выполнение и других работ. Зачем рисковать и нарываться на то или другое, когда лучше немного побродить по сети и сдать работу в срок?

Пять дней, отведенные в календарном графике на работу, — это, по существу, разрешение разработчику использовать именно столько на выполнение задания. Человеку свойственно при опережении графика заполнять сэкономленное время другими, более интересными для него занятиями.

Запаздывание распространяется на последующие этапы календарного графика

Поскольку традиционные планы составляются на основе видов деятельности, они по большому счету сфокусированы на взаимозависимости работ. Рассмотрим диаграмму Ганта (рис. 2.1), где представлены четыре вида деятельности и их взаимозависимости.

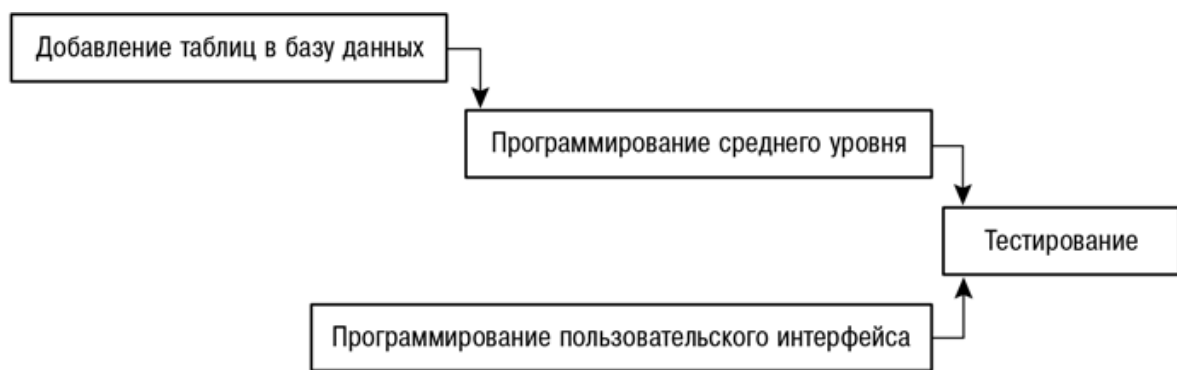


Рис. 2.1. Тестирование начнется с задержкой, если хоть что-нибудь пойдет хуже запланированного; оно начнется досрочно лишь в том случае, если всё без исключения будет лучше запланированного

Для досрочного начала тестирования требуется совпадение следующих событий:

- досрочное завершение программирования среднего уровня, которое зависит от срока завершения добавления таблиц в базу данных;
- досрочное завершение программирования пользовательского интерфейса;
- досрочное высвобождение тестировщика.

Ключевым моментом является то, что даже в этом простом случае досрочное начало тестирования зависит от выполнения всех трех условий. В то же время если для досрочного начала тестирования необходимо выполнение целого ряда условий, то для задержки тестирования достаточно наступления любого из перечисленных ниже событий:

- задержка завершения программирования пользовательского интерфейса;
- программирование среднего уровня требует больше времени, чем планировалось, и завершается позже;
- программирование среднего уровня укладывается в отведенное планом время, но начинается позже из-за задержки добавления таблиц в базу данных;
- недоступность тестировщика.

Другими словами, для досрочного начала необходимо сочетание условий, а для задержки начала достаточно одной причины.

Проблема осложняется тем, что, как мы уже говорили, работы очень редко завершаются досрочно. Это означает, что они обычно начинаются с опозданием и что запаздывание распространяется на последующие этапы календарного графика. Поскольку досрочное завершение — явление редкое, такой вид деятельности, как тестирование на рис. 2.1, начинается досрочно

еще реже.

Работы не являются независимыми

Считается, что работы не зависят друг от друга, если сроки исполнения одной из них не влияют на сроки исполнения другой. При строительстве дома время подготовки котлована для фундамента не зависит от времени, необходимого для покраски стен. Когда работы не зависят друг от друга, задержку окончания одной из них можно компенсировать досрочным завершением другой. Многократное подбрасывание монеты — другой пример независимых видов деятельности. Если при первом подбрасывании выпадает орел, то это никак не влияет на вероятность выпадения орла при втором подбрасывании.

Являются ли работы, производимые в процессе разработки программного обеспечения, независимыми? Могут ли вариации сроков их завершения компенсировать друг друга? К сожалению, нет. Многие виды деятельности, связанные с разработкой программного обеспечения, нельзя считать независимыми. Например, если я пишу клиентскую часть приложения и первый экран отнимает на 50% больше времени, чем запланировано, высока вероятность того, что каждый из оставшихся экранов также потребует больше времени. Если операции процесса разработки не являются независимыми, то вариации сроков их завершения не компенсируют друг друга.

В типичном плане проекта многие работы не являются независимыми, однако мы снова и снова забываем об этом. Когда кто-то задерживает сдачу первого из нескольких сходных элементов, мы слышим такое оправдание: «Да, я запоздал в этот раз, но дальше отставание будет наверстано». Это следствие надежды на то, что опыт, полученный при выполнении первой работы, позволит завершить оставшиеся работы раньше, чем предусмотрено планом. В реальности же подобная ситуация должна говорить нам, что, если какая-то работа занимает больше времени, чем запланировано, все остальные сходные работы тоже, скорее всего, потребуют больше времени.

Многозадачность приводит к дальнейшим задержкам

Второй причиной неудовлетворительных результатов традиционных подходов к планированию является многозадачность, под которой понимается одновременное выполнение нескольких задач. Многозадачность ужасным образом сказывается на производительности.

Кларк и Уилрайт (Clark and Wheelwright, 1993) в своем исследовании эффектов многозадачности пришли к выводу, что время, посвящаемое создающей стоимости работе, быстро сокращается, когда человек занимается более чем двумя задачами. Этот эффект виден на рис. 2.2, где представлены результаты этого исследования.

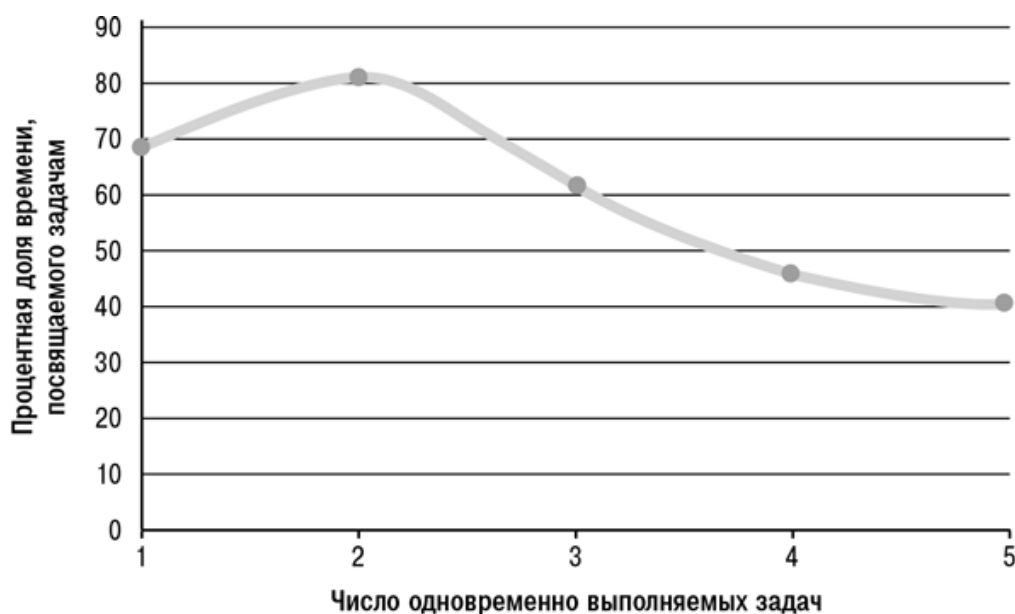


Рис. 2.2. Влияние многозадачности на производительность

По логике следует, что многозадачность помогает, когда вы занимаетесь двумя вещами, — если выполнение одной из них стопорится, вы можете переключиться на другую. Логично и показанное на рис. 2.2 быстрое сокращение времени, посвящаемого создающим стоимостью задачам, когда их становится больше двух. Редко когда застопоривается более чем одна задача за раз, а если мы работаем над тремя и более задачами одновременно, время на переключение с одной из них на другую оборачивается более ощутимыми затратами и бременем.

Многозадачность нередко превращается в проблему, когда какие-либо проектные работы начинают завершаться с запозданием. В этом случае взаимозависимость между видами работ становится критически важной. Разработчик, ожидающий завершения задачи своим коллегой, начинает просить последнего предоставить ему хотя бы сокращенную версию, чтобы можно было продолжить работу. Допустим, мне отведено 10 дней на работу с определенными изменениями базы данных, потом 10 дней на реализацию интерфейса прикладной программы (ИПП) для доступа к базе данных, а затем 10 дней на разработку пользовательского интерфейса. Эта ситуация отражена в верхней части рис. 2.3. Ваша работа не может начаться до тех пор, пока вы не получите ИПП от меня. Вы просите меня сделать необходимый минимум работы по ИПП, чтобы начать выполнение своей

задачи. Аналогичным образом тестировщик просит меня сделать минимальную версию пользовательского интерфейса, чтобы он мог начать тестирование. Я соглашаюсь, и мой календарный график приобретает вид, представленный в нижней части рис. 2.3.



Рис. 2.3. Многозадачность оттягивает срок завершения работы и увеличивает продолжительность существования незавершенных работ

Это зачастую создает иллюзию скорости, однако, как видно на рис. 2.3, моя работа над базой данных и ИПП завершается позже, чем первоначально планировалось. Вряд ли стоит сомневаться в том, что это повлияет на последующие запланированные работы. Кроме того, в нашем примере каждый из затребованных видов работ остается незавершенным в течение 20, а не 10 дней, которые потребовались бы при последовательном выполнении работ.

Ситуацию усугубляет то, что рис. 2.3 не предполагает замедления исполнения работ в результате более частого переключения между ними. Кларк и Уилрайт показывают, что производительность снижается.

Многозадачность превращается в проблему при традиционном планировании проекта по двум основным причинам. Во-первых, работы обычно закладывают в план задолго до их начала, а эффективно распределить работы заранее невозможно. Закрепление работы за конкретным исполнителем, а не за группой углубляет проблему. Во-вторых, многозадачность заставляет фокусироваться на высоком уровне загрузки всех исполнителей в проекте, а не на создании необходимого резерва, позволяющего справиться с неизбежной изменчивостью типичных задач проекта. Загрузка всех на 100% приводит к такому же результату, как и загрузка скоростного шоссе на 100%, — движение останавливается и никто не может стронуться с места.

Функции не разрабатываются в

соответствии с их приоритетом

Третья причина, по которой традиционное планирование не может обеспечить стабильное создание высокоценных продуктов, заключается в том, что работы, описанные в плане, не приоритизируются по их ценности для пользователей и клиента. Многие традиционные планы составляются в предположении, что все без исключения идентифицированные виды работ должны быть выполнены. Это означает, что работы обычно ранжируются и выстраиваются в определенной последовательности для удобства команды разработчиков.

В соответствии с традиционным мышлением если выполнению подлежат все виды работ, то для клиентов проекта не имеет значения, в какой последовательности они выполняются. Такой подход приводит к тому, что команда разработчиков занимается созданием функций в случайном, с точки зрения клиента, порядке. Затем в конце проекта команда, пытаясь уложиться в календарный график, начинает сокращать набор функций. Поскольку никто не старается выстроить работу над функциями в зависимости от их приоритетности, среди отброшенных функций оказываются такие, которые имеют более значительную ценность, чем функции, включенные в продукт.

Мы не учитываем неопределенность

Четвертый изъян традиционных подходов к планированию — игнорирование неопределенности. Мы не принимаем во внимание неопределенность, связанную с продуктом, и считаем, что первоначальный анализ требований позволяет определить полный и идеальный набор характеристик продукта. Мы исходим из того, что пользователи не передумают, не изменят свое мнение и не выдвинут новые требования в период, охватываемый планом.

Аналогичным образом мы не учитываем неопределенность, связанную с процессом создания продукта, и делаем вид, что можем дать точную оценку («две недели») работе, которой присуща неопределенность. Невозможно идентифицировать каждое действие, которое потребуется для осуществления проекта. Так или иначе, мы нередко не учитываем этот аспект в составляемых планах.

Несмотря на неопределенность, календарные графики зачастую содержат конкретные, безусловные даты, например «Поставка продукта — 30 июня». В начале проекта неопределенность наиболее высока. Оценки, которые мы даем, должны отражать эту неопределенность. Можно,

например, представить срок окончания в виде диапазона: «Поставка продукта — июнь–август». Оценки по мере выполнения проекта и устранения неопределенности и риска можно пересматривать и уточнять. Именно в этом суть конуса неопределенности, представленного в главе 1 «Цель планирования».

Справиться с неопределенностью лучше всего помогает итеративный подход. Чтобы снизить неопределенность, связанную с конечным обликом продукта, разбейте процесс выполнения работы на короткие итерации и показывайте (или в идеале предоставляйте) пользователям работоспособные версии программного обеспечения каждые несколько недель. Неопределенность, связанная с тем, как разрабатывать продукт, также устраняется путем итераций. Это позволяет, например, включить в план упущенные задачи, скорректировать допущенные ошибки. При таком подходе фокус смещается с плана на планирование.

Оценки превращаются в обязательства

Любая оценка — это вероятностная величина, предполагающая, что работа будет выполнена в расчетный срок. Допустим, вашей команде дают задание разработать новый высокоэффективный текстовый процессор. Вероятность завершения этой работы к концу недели равна 0%. Вероятность ее выполнения через 10 лет равна 100%. Если я попрошу вас дать оценку срока и вы назовете конец недели, то вероятность ее реализации будет нулевой. Если вы назовете 10 лет, то вероятность реализации оценки будет 100%-ной. Вероятность реализации всех оценок в диапазоне от конца недели до 10 лет будет находиться в интервале от 0 до 100% (Armour, 2002).

При традиционном подходе проблема может возникнуть, если проектная команда или другие участники проекта будут смешивать оценку с принятием обязательств. Как подчеркивает Филип Армор (Armour, 2002), оценка — это вероятностная величина, а обязательство не может быть вероятностным. Обязательства принимаются в виде конкретных дат. Обычно дата, которую команде предлагают (или требуют) принять в качестве обязательства, имеет, с ее точки зрения, менее чем 100%-ную вероятность. Прежде чем принять такое обязательство, команде необходимо учесть целый ряд экономических факторов и рисков. Очень важно, чтобы у команды была такая возможность и чтобы каждая оценка не превращалась автоматически в обязательство.

Резюме

С учетом представленного выше перечня проблем, связанных с традиционными подходами к планированию, вряд ли можно удивляться неудовлетворительным результатам многих проектов. Процессно-ориентированное планирование отвлекает внимание от функций, которые на деле определяют потребительскую стоимость продукта для клиента. Помимо этого, целый ряд проблем возникает в связи с вероятностью задержки поставок относительно календарного графика, полученного на основе процессно-ориентированного планирования. С самыми благими намерениями участники проектов уповают на многозадачность как возможное решение проблем, однако в конечном итоге еще больше отстают от календарного графика из-за скрытых издержек многозадачности. Когда не удастся выдержать график выполнения проекта, исполнители начинают урезать функциональность. Поскольку работа над функциями ведется в том порядке, который наиболее удобен разработчикам, далеко не всегда в жертву приносятся функции, имеющие наименьшую потребительскую стоимость для пользователей.

Игнорирование неопределенности конечных требований пользователей может приводить к завершению проекта в срок, однако без включения в него значимых возможностей, которые были идентифицированы после принятия плана. Кроме того, игнорирование неопределенности подходов к разработке продукта связано с риском упустить те или иные задачи и не включить их в план проекта. Это, в свою очередь, повышает вероятность задержки проекта, урезания функциональности или непереносимого снижения качества продукта.

Многие организации путают оценки с обязательствами. Как только команда предоставляет оценку, ее тут же превращают в обязательство.

Вопросы для обсуждения

1. Какие проблемы возникают, если планы составляются на основе видов деятельности, а не на основе поставляемых функций?
2. Существует ли у вас практика отождествления оценки и обязательства? Какие проблемы это влечет? Что можно сделать для изменения ситуации?
3. Как многозадачность влияет на ваш текущий проект? Каким образом можно уменьшить ее влияние?

Глава 3

Agile-подход

Хороший план, составленный сегодня, лучше идеального плана, который появится на следующей неделе.

Генерал Джордж Паттон

Хотя этот процесс начался намного раньше, официально agile-движение существует с момента принятия Agile-манифеста в феврале 2001 г. (Beck et al.). Манифест был разработан и подписан 17 «идеологами облегченных методологий», как они называли себя в то время. Их документ дал и название проповедуемому ими подходу к разработке программного обеспечения, и заявления о ценностях. Авторы Agile-манифеста писали о том, что для них более значительную ценность имеют:

- люди и взаимодействия, а не процессы и инструменты;
- работающая программа, а не полный пакет документации;
- сотрудничество с клиентом, а не переговоры по условиям контракта;
- реагирование на изменение, а не следование плану.

Agile-команды считают людей и взаимодействия более ценными, чем процессы и инструменты, в силу уверенности в том, что слаженно работающая команда высокопрофессиональных исполнителей с посредственными инструментами при любых обстоятельствах превосходит неработоспособную команду посредственных исполнителей с превосходными инструментами и процессами. Выдающиеся программы создаются выдающимися людьми, и как отрасль мы слишком долго и безуспешно занимались созданием процесса разработки, в котором людям отводится роль взаимозаменяемых винтиков в механизме. Agile-процессы строятся на признании уникальных способностей (и недостатков) отдельных людей и их использовании для извлечения выгоды вместо попыток сделать всех одинаковыми.

Agile-команды считают работающую программу более ценной, чем полный пакет документации, потому что она ведет к получению стабильной, последовательно улучшаемой версии продукта в конце каждой итерации. Такой подход позволяет быстро и часто получать обратную связь

от потребителей относительно продукта и процесса. По мере улучшения разработанной программы с каждой итерацией ее можно демонстрировать потенциальным или реальным пользователям. Их отзывы и замечания учитываются в процессе разработки, что позволяет команде всегда работать над наиболее ценными функциями и гарантирует удовлетворение ожиданий пользователей.

Сотрудничество с клиентом ценится выше переговоров по условиям контракта, потому что agile-команды предпочитают вовлекать в работу над общими целями все стороны проекта. Переговоры по условиям контракта иногда заставляют команду разработчиков и клиента проекта занимать непримиримые позиции с самого начала. Мне нравятся игры, и, когда моей старшей дочери исполнилось четыре, я подарил ей «кооперативную игру», поскольку она, на мой взгляд, должна была понравиться и поскольку я понятия не имел, насколько увлекательны кооперативные игры. В купленной мною игре на принцессу было наложено заклятие и игрокам нужно было преодолевать препятствия (ров, наполненный водой, запертая дверь и т.п.), чтобы добраться до принцессы. Игроки делали ходы по очереди, как и в большинстве игр, однако цель заключалась в преодолении препятствий сообща и спасении принцессы. Все либо выигрывали, либо проигрывали. Эта игра очень увлекательна, и нам хотелось бы, чтобы, как и в ней, команды разработчиков программного обеспечения и клиенты подходили к проектам с желанием сотрудничать и двигаться к общим целям. Конечно, без контрактов зачастую не обойтись, однако от контрактных условий и деталей очень сильно зависит, как будут взаимодействовать стороны проекта — на основе сотрудничества или на основе соперничества.

Agile-команды считают реагирование на изменение более ценным, чем следование плану, потому что они заинтересованы в поставке максимально возможной потребительской стоимости клиенту проекта и пользователям. Пользователи никогда, кроме самых простых проектов, заранее не знают во всех деталях, какие именно функции им нужны. Поэтому они неизбежно выходят с новыми идеями, и почти так же неизбежно некоторые функции, которые они считают необходимыми сегодня, оказываются низкоприоритетными завтра. Для agile-команды план является лишь одним из целого ряда возможных представлений о будущем. По мере того как команда приобретает знания и опыт, она учитывает их в плане. Команда может продвигаться в разработке быстрее или медленнее, чем первоначально ожидалось. Не исключено, что какой-то конкретный набор функций понравится пользователям больше, чем ожидалось, а какая-то функция, которая первоначально считалась критически важной, полностью разонравится.

Итак, с учетом четырех заявлений о ценностях Agile-манифеста

рассмотрим, что понимается под agile-подходом к проекту и что такое agile-подход к оценке и планированию.

Agile-подход к проекту

Теперь, имея представление о четырех основополагающих заявлениях по agile-ценностям, посмотрим, что представляет собой agile-команда на практике. Четыре заявления по ценностям, взятые в целом, приводят к итеративным и последовательным процессам разработки программного обеспечения, которые позволяют получать работоспособную и протестированную программу в конце каждой итерации. Последующие разделы посвящены основным аспектам работы agile-команд, в том числе:

- работа единой командой;
- работа короткими итерациями;
- поставка какого-либо результата после каждой итерации;
- фокус на бизнес-приоритетах;
- проверка и модифицирование.

Agile-команда работает как единое целое

Критически важно для успеха проекта, чтобы все участники считали себя членами одной команды, имеющей общую цель. В agile-проекте нет места менталитету «самоустранение от участия в дальнейшем процессе после выполнения своей непосредственной задачи». Аналитики не уходят в тень после выдачи требований дизайнерам. Дизайнеры и системные архитекторы не отстраняются от работы после выдачи заданий программистам, а программисты не бросают без поддержки тестировщиков. Успешной agile-команде необходимо мышление «мы все работаем над этим вместе». Хотя agile-команда должна работать как единое целое, в ней есть целый ряд конкретных ролей. Полезно понимать, что это за роли и каково их место в agile-подходе к оценке и планированию.

Первая роль — *владелец продукта*. В число основных обязанностей владельца продукта входит формирование общего видения проекта у всех членов команды, определение приоритетов, обеспечивающих разработку наиболее ценной функциональности в первую очередь, а также принятие решений, направленных на получение хорошей рентабельности инвестиций в проект. При разработке коммерческого программного обеспечения владелец продукта нередко является представителем службы маркетинга или управления проектами компании. Если программное обеспечение разрабатывается для внутреннего использования, то владелец продукта

может быть пользователем, руководителем пользователей, аналитиком или лицом, финансирующим проект.

Вторая роль — *клиент*. Клиент — это лицо, которое принимает решение о финансировании проекта или о покупке программы. В проекте по разработке программного обеспечения для внутреннего пользования клиент обычно является представителем другой группы или подразделения. В таких проектах роли владельца продукта и клиента нередко совмещаются. В случае продукта, распространяемого на коммерческой основе, в роли клиента выступает лицо, которое покупает программу. В обеих ситуациях клиент может с равным успехом быть *пользователем* программного обеспечения или не быть им, и это самостоятельная важная роль.

Еще одна роль, которую следует отметить, — *разработчик*. Я использую понятие «разработчик» в очень широком смысле, подразумевая любого разработчика программного обеспечения. В их число входят программисты, тестировщики, аналитики, администраторы баз данных, юзабилити-эксперты, технические писатели, системные архитекторы, дизайнеры и т.д. При таком подходе к использованию данного термина даже владелец продукта может считаться разработчиком ряда проектов.

Последняя роль — *руководитель проекта*. Как пишет Хайсмит (Highsmith, 2004a), роль руководителя проекта изменяется в случае применения agile-подхода. Руководители agile-проектов концентрируют внимание больше на лидерстве, а не на менеджменте. В некоторых agile-проектах лицо, выполняющее роль руководителя проекта, выступает также и в другой роли: нередко как разработчик, а иногда как владелец продукта.

Работа agile-команд разбивается на короткие итерации

При agile-подходе к выполнению проекта нет общей разбивки работы на этапы — нет этапа формулирования предварительных требований, за которым следует анализ, разработка архитектуры и т.д. В зависимости от фактически выбранного или определенного вами agile-процесса можно перед началом проекта предусмотреть очень короткий этап проектирования, моделирования или чего-либо в этом роде. Однако, как только начинается реальное осуществление проекта, все работы (анализ, дизайн, программирование, тестирование и т.п.) выполняются параллельно на каждой итерации.

Итерации *ограничиваются по времени*, т.е. они завершаются вовремя, даже если приходится урезать функциональность. Временные рамки нередко очень короткие. Большинство agile-команд работают итерациями продолжительностью от двух до четырех недель, однако бывают случаи,

когда итерации увеличиваются до трех месяцев. Большинство команд выбирают сравнительно постоянную длину итераций, вместе с тем существует также практика определения подходящей длины перед началом каждой итерации.

Agile-команда поставляет какой-либо результат после каждой итерации

Более важным, чем выбор определенной длины итерации, является то, что в процессе итерации команда превращает одно или несколько неточно сформулированных требований в скомпонованную, протестированную и потенциально готовую к поставке программу. Конечно, многие команды не поставляют результаты каждой итерации пользователям — цель заключается в том, чтобы это в принципе можно было сделать. Это означает, что команда последовательно добавляет одну или несколько небольших функций во время каждой итерации, но каждая добавленная функция встроена в продукт, протестирована и имеет качество, необходимое для релиза.

Принципиально важно, чтобы к концу каждой итерации продукт находился в потенциально готовом к поставке состоянии. На практике это не означает, что команда должна сделать абсолютно все необходимое для выпуска продукта, — в конце концов, релизы необязательно выпускаются после каждой итерации. Например, я работаю с командой, которой нужно два месяца на тестирование среднего времени наработки на отказ, включающее и аппаратную часть, и программное обеспечение. Она не может сократить этот срок, поскольку он оговорен клиентом в контракте и именно такое время зачастую необходимо для выявления сбоев аппаратной части. Команда работает с четырехнедельными итерациями, и, помимо тестирования среднего времени наработки на отказ, она доводит свой продукт до работоспособного состояния в конце каждой итерации.

Поскольку одной итерации обычно недостаточно по времени для включения новой функциональности, удовлетворяющей потребности пользователя или клиента, вводится более широкая концепция *релиза*. Релиз содержит одну или несколько (обычно несколько) итераций, которые последовательно дополняют друг друга, давая полный набор соответствующих функций. Если итерации чаще всего длятся от двух до четырех недель, то на релиз обычно требуется от двух до шести месяцев. Например, в системе управления инвестициями один релиз может включать все функции, связанные с покупкой и продажей взаимных фондов и фондов денежного рынка. На завершение такого релиза могут потребоваться шесть двухнедельных итераций (примерно три месяца). Второй релиз может

добавить торговлю акциями и облигациями и потребовать четыре дополнительные двухнедельные итерации. Релизы могут выпускаться с разными интервалами. Скажем, на разработку первого релиза необходимо шесть месяцев, следующего релиза — три месяца и т.д.

Agile-команда фокусируется на бизнес-приоритетах

Agile-команды демонстрируют приверженность бизнес-приоритетам двумя путями. Во-первых, они поставляют функции в порядке, определенном владельцем продукта, который должен ранжировать и объединять функции в релиз таким образом, чтобы оптимизировать рентабельность инвестиций организации в проект. С этой целью составляется план релиза на основе возможностей команды и ранжированного по приоритетности перечня желаемых новых функций. Для обеспечения владельцу продукта максимальной гибкости приоритизации функции необходимо описать так, чтобы минимизировать их техническую взаимозависимость. Владелец продукта сложно ранжировать функции по приоритетности в плане релиза, если выбор одной функции требует предварительной разработки трех других. Команде вряд ли удастся полностью исключить взаимозависимости, однако во многих случаях достаточно свести их к минимуму.

Во-вторых, agile-команды фокусируются на разработке и поставке ценных для пользователей функций, а не на выполнении изолированных задач (которые в конечном итоге объединяются в ценную для пользователей функцию). Одним из лучших подходов к этому является работа с пользовательскими историями, которая представляет собой облегченную методологию представления требований к программному обеспечению (Cohn, 2004). *Пользовательская история* — это краткое описание функциональности с точки зрения пользователя или клиента системы. Пользовательские истории излагаются в свободной форме, какие-либо синтаксические правила их составления отсутствуют. Тем не менее в целом неплохо придерживаться следующей формы: «Как <тип пользователя> я хочу <иметь возможность> с тем, чтобы <деловая ценность>». Воспользовавшись этим шаблоном, можно написать, например, такую историю: «Как покупатель книг я хочу осуществлять их поиск по номеру ISBN с тем, чтобы быстро отыскивать нужное издание».

Пользовательские истории являются облегченной методологией, потому что работа по их сбору и документированию не выполняется заранее. Вместо составления объемных спецификаций с описанием требований agile-команды предпочитают использовать подход «точно вовремя». Обычно он начинается с короткого описания пользовательской истории на карточке или, возможно, в компьютере (в случае большой и распределенной команды). Впрочем, карточка с историей — это всего лишь отправная

точка, и каждая пользовательская история сопровождается диалогами между разработчиками и владельцем продукта. Такие диалоги происходят по мере необходимости, и в них участвуют все, кому это требуется. Использование подхода на основе пользовательских историй не мешает существованию обычной письменной документации. Вместе с тем фокус кардинальным образом смещается с письменного общения в сторону устного.

Agile-команда проверяет и модифицирует

План, принятый в начале проекта, не является гарантией того, что произойдет далее. По существу, это всего лишь существующее в данный момент предположение о том, что произойдет. Против плана работает множество событий — исполнители могут присоединяться к проекту или уходить из него, технологии могут работать лучше или хуже ожидаемого, пользователи могут менять свое мнение, конкуренты могут заставить нас реагировать иным образом или действовать быстрее и т.д. Agile-команды воспринимают любое такое изменение как возможность и необходимость обновления плана, с тем чтобы он лучше отражал реалии текущей ситуации.

В начале каждой итерации agile-команда учитывает все новые знания, полученные на предыдущей итерации, и соответствующим образом модифицирует проект. Если команда получает информацию о чем-то таком, что может повлиять на точность или ценность плана, она корректирует план. На точности плана может сказаться, например, переоценка или недооценка скорости продвижения команды. Или может оказаться, что определенный вид работ требует больше времени, чем предполагалось раньше.

Ценность плана может измениться в результате получения владельцем продукта определенных знаний о потребностях вероятных пользователей. Возможно, из отзывов пользователей о программе, полученной в предыдущей итерации, владелец продукта узнает, что одну функцию они хотели бы расширить, а другая функция не заинтересовала их в той мере, в какой предполагалось. Ценность плана в этом случае может быть повышена в результате включения большего числа предпочтительных функций в релиз за счет исключения некоторых менее ценных функций.

Ничто из сказанного не означает, что agile-команды бессистемно подходят к изменению приоритетов. Приоритеты обычно довольно стабильно переходят от одной итерации к другой. Однако возможность изменения приоритетов между итерациями служит мощным инструментом максимизации рентабельности инвестиций в проект.

Agile-подход к планированию

И без того трудная задача оценки и планирования разработки нового продукта дополнительно осложняется нашими ошибочными представлениями о проектах. Макомбер (Macomber, 2004) подчеркивает, что ни в коем случае нельзя рассматривать проект исключительно как выполнение ряда последовательных этапов. На проект необходимо смотреть как на быстрое и стабильное генерирование потока новых полезных возможностей и новых знаний. Новые возможности встраиваются в продукт, новые знания используются для его совершенствования.

В agile-проекте этот поток новых возможностей и знаний служит основой для управления текущей работой. Новые знания, генерируемые в процессе выполнения проекта, могут относиться к продукту или к проекту. Новые знания о продукте помогают нам получать больше информации о том, каким должен быть продукт. Новые знания о проекте — это информация о команде, используемых технологиях, рисках и т.п.

Мы зачастую не учитываем эти новые знания и не планируем возможность их получения. Как результат, процесс планирования строится на допущении о том, что нам уже известно все необходимое для получения точного плана. В мире разработки программного обеспечения такое случается редко, если вообще случается. Уорд Каннигем заметил, что «это больше планирование того, что вы хотите узнать, а не того, что [продукт] получится в конце» (Van Schooenderwoert, 2004).

Я часто сравниваю традиционный взгляд на проект с 10-километровым забегом. Вы точно знаете, как далеко находится финишная черта, и ваша цель заключается в том, чтобы ее достичь максимально быстро. В agile-проекте мы не знаем точно, далеко ли финиш, но нередко известно, что достичь его нужно как можно ближе к известной дате. Agile-проект больше похож на гонку по времени, а не на забег на 10 км: вам нужно пробежать как можно больше за 60 минут. Иначе говоря, команда agile-проекта знает, когда финиширует, но не знает, какой результат покажет. Когда мы признаем, что результат — это нечто неизвестное и не поддающееся выяснению заранее, планирование становится процессом определения и пересмотра задач, который ведет к достижению долгосрочной цели.

Многоуровневость планирования

При постановке задач и их пересмотре важно помнить, что мы не можем видеть дальше горизонта и что точность плана быстро снижается по мере того, как мы все дальше уходим за черту, до которой видим. Допустим, вы стоите на палубе небольшого судна и ваши глаза находятся на высоте 2,7 м

над уровнем воды. Расстояние до горизонта в этом случае составляет примерно 6 км[2]. Если вы планируете 30-километровое путешествие, то вам необходимо составить план на перспективу как минимум в пять раз дальше горизонта, который составляет 6 км. Поскольку вы не можете видеть дальше горизонта, нужно периодически осматриваться и корректировать свой план.

Проект оказывается в зоне риска, если планирование выходит за пределы горизонта составителя плана и не предусматривает возможности осмотреться, определить новый горизонт и внести изменения. Необходима последовательная разработка плана. Agile-команды достигают этого, осуществляя планирование для трех четко определенных горизонтов — релиз, итерация и текущий день. Взаимосвязь между этими (и другими) горизонтами планирования показана в виде так называемой луковицы планирования на рис. 3.1.

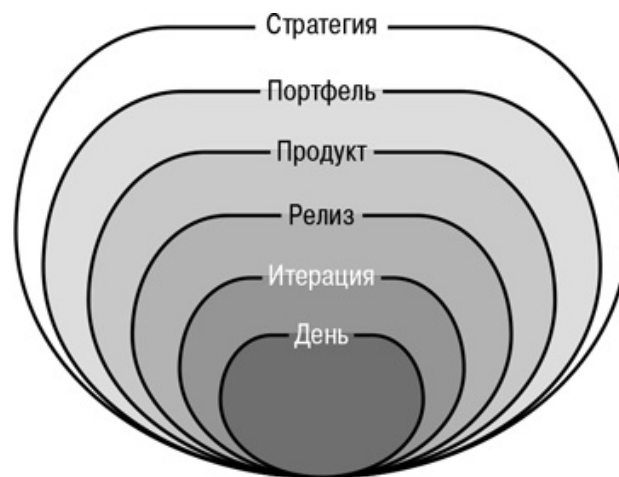


Рис. 3.1. Луковица планирования; agile-команды составляют планы как минимум на уровне релиза, итерации и текущего дня

Большинство agile-команд интересуют только три наиболее низких уровня луковицы планирования. При планировании релиза учитываются пользовательские истории или темы, которые создаются для нового релиза продукта или системы. Цель планирования релиза — это поиск приемлемых ответов на вопросы об объеме, календарном графике и ресурсах для проекта. Планирование релиза осуществляется в начале проекта, однако это не разовое действие. Хороший план релиза обновляется периодически на протяжении всего проекта (обычно в начале каждой итерации), с тем чтобы он всегда отражал текущие ожидания относительно включаемой в релиз функциональности.

Следующий уровень — планирование итерации, осуществляемое в начале каждой из них. Опираясь на результаты только что завершенной итерации, владелец продукта идентифицирует высокоприоритетную работу,

которой команда должна заниматься в новой итерации. Поскольку на этом уровне мы имеем более близкий горизонт, чем при планировании релиза, компоненты плана итерации могут быть более мелкими. В процессе планирования итерации мы говорим о задачах, реализация которых необходима для превращения запроса на функцию в работающую и протестированную программу.

Наконец, существует еще дневное планирование. Большинство agile-команд ежедневно проводят своего рода летучки для координирования работы и синхронизации действий. Хотя с формальной точки зрения такое планирование может показаться излишним, на этих летучках команды действительно составляют, оценивают и пересматривают свои планы. Во время ежедневных совещаний команды ограничивают горизонт планирования следующим днем, когда они вновь соберутся для обсуждения дел, поэтому они фокусируются на планировании задач и на координировании отдельных видов деятельности, необходимых для выполнения этих задач.

Осуществляя планирование на трех временных горизонтах — релиз, итерация и день, — agile-команды концентрируют внимание на видимых и важных для создаваемого плана аспектах.

Планирование на уровне продукта и портфеля, а также стратегическое планирование находятся вне сферы интереса большинства agile-команд (и настоящей книги). Планирование на уровне продукта требует от владельца продукта прогнозирования ситуации на более далеком горизонте, чем выпуск ближайшего релиза, и планирования эволюции выпущенного продукта или системы. Планирование на уровне портфеля предполагает выбор продуктов, которые наилучшим образом соответствуют видению, сформированному в процессе стратегического планирования организации.

Состояние удовлетворенности

Любой проект начинается с набора целей. Ваш текущий проект может быть нацелен на создание лучшего в мире текстового процессора. Однако цели проекта, как правило, этим не исчерпываются. Наверняка существуют дополнительные цели, связанные с календарным графиком, бюджетом и качеством. Эти цели можно считать *условиями удовлетворенности* клиента или владельца продукта — иначе говоря, критериями, которые используются для оценки успешности проекта.

Когда-то, во времена моей учебы в средней школе, получив задание написать реферат по какой-нибудь книге, скажем «Моби Дик», я всегда спрашивал учительницу, какого объема должен быть этот реферат. Учительница говорила, например, «пять страниц», и я получал информацию о ее первом условии удовлетворенности. Конечно, существовал еще ряд

дополнительных, неписанных условий удовлетворенности. Например, подразумевалось, что реферат должен быть написан аккуратно, что он написан мною лично, на английском языке и т.д.

В начале планирования релиза команда и владелец продукта совместно обговаривают условия удовлетворенности владельца продукта. Они включают в себя обычные аспекты — объем, календарный график, бюджет и качество, — хотя agile-команды обычно предпочитают рассматривать качество как безусловный показатель. Команда и владелец продукта определяют пути выполнения всех условий удовлетворенности. Владелец продукта может, например, в равной мере удовлетворять выпуск через пять месяцев релиза, включающего определенный набор пользовательских историй, и выпуск на месяц позже релиза, включающего дополнительные пользовательские истории.

Случается, однако, что все условия удовлетворенности владельца продукта выполнить невозможно. Команда может создать лучший в мире текстовый процессор, но у нее нет возможностей сделать это к следующему месяцу. Если приемлемое решение найти не удастся, условия удовлетворенности необходимо менять. По этой причине планирование релиза и выяснение условий удовлетворенности владельца продукта являются итеративными процессами, как показано на рис. 3.2.

После составления плана релиза, охватывающего примерно следующие три–шесть месяцев, его используют как основу для планирования первой итерации. Аналогично планированию релиза планирование итерации начинается с рассмотрения условий удовлетворенности владельца продукта. Для итерации эти условия обычно включают в себя функции, которые он хотел бы получить, а также высокоуровневое тестирование этих функций.

В качестве примера рассмотрим туристический сайт, который включает такую пользовательскую историю: «Как пользователь я хочу иметь возможность аннулирования заказа». При обсуждении этой истории с владельцем продукта разработчики узнают, что его условия удовлетворенности включают в себя следующее:

- Пользователь, который аннулирует заказ более чем за 24 часа, получает полное возмещение своих средств.
- Пользователь, который аннулирует заказ менее чем за 24 часа, получает возмещение своих средств за вычетом комиссии в размере \$25.
- Условия аннулирования заказа размещаются на сайте и высылаются пользователю по электронной почте.

Как и планирование релиза, планирование итерации является итеративным процессом. Владелец продукта и команда обсуждают различные пути достижения условий удовлетворенности для итерации.

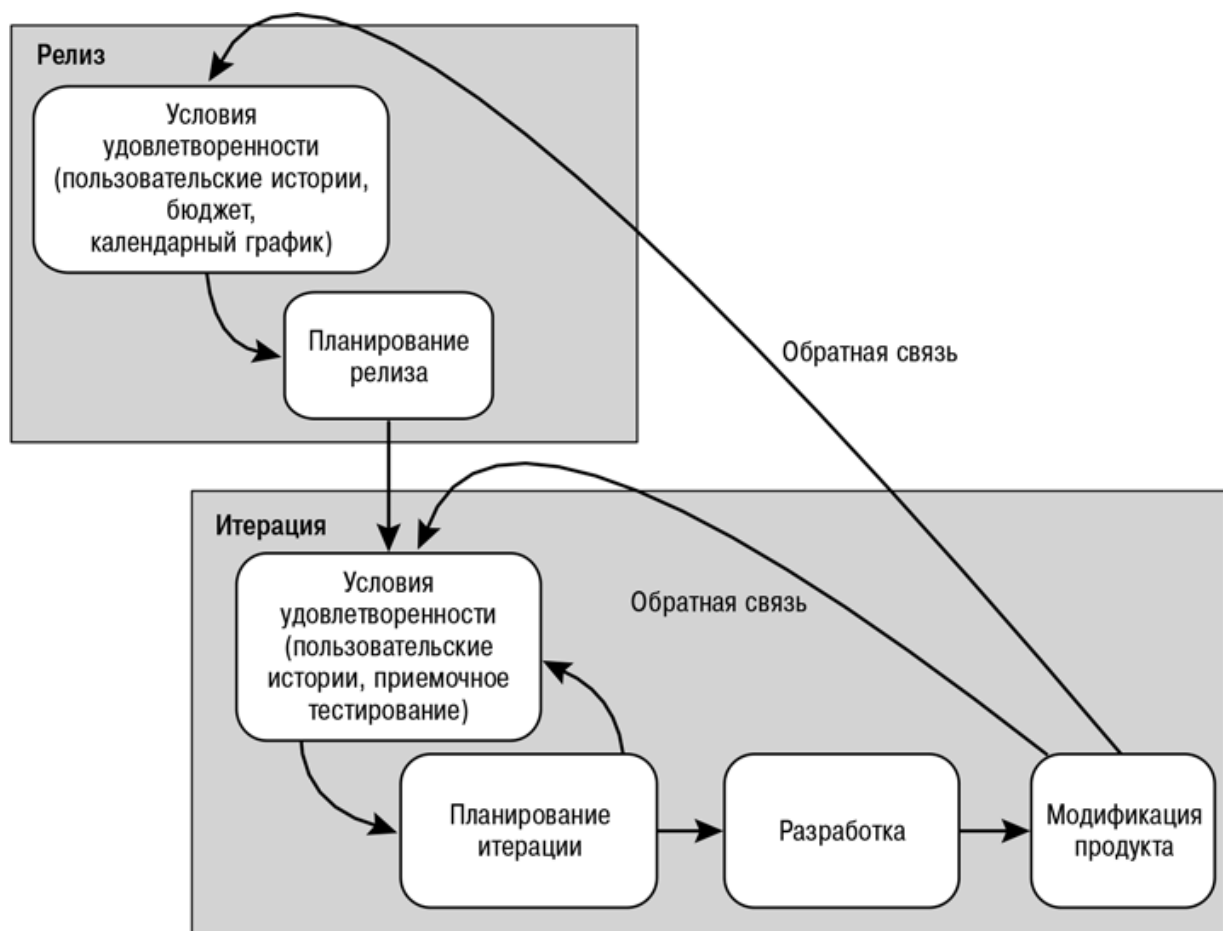


Рис. 3.2. Условия удовлетворенности определяют ход планирования релиза и итерации

На рис. 3.2 показана петля обратной связи от новой модификации продукта к этапу определения условий удовлетворенности в начале обоих процессов планирования релиза и итерации. В результате опыта, полученного при создании модификации продукта в процессе итерации, команда может приобрести знания, влияющие на планирование на одном или нескольких уровнях. Аналогичным образом, представляя модификацию продукта существующим или потенциальным пользователям, можно получать новые знания, заставляющие вносить изменения в планы. Agile-команда встраивает эти изменения в планы в той мере, в какой это позволяет создавать более ценный продукт.

Резюме

Agile-команды работают как единое целое, однако в них существует целый ряд конкретных ролей. Во-первых, владелец продукта, который отвечает за формирование общего видения проекта и за определение очередности разработки функций. Во-вторых, клиент, который принимает решение о финансировании проекта или о покупке программы после ее разработки.

Помимо этих ролей в agile-проекте есть еще пользователи, разработчики и менеджеры.

Работа agile-команды разбивается на короткие, ограниченные по времени итерации, и каждая из них завершается поставкой работоспособного продукта. Функции, разрабатываемые в процессе выполнения итераций, выбираются на основе их приоритетности для бизнеса. Это позволяет гарантировать первоочередную разработку наиболее важных функций. Пользовательские истории — наиболее распространенный подход, применяемый agile-командами для представления потребностей пользователей. Agile-команды исходят из того, что планы быстро устаревают. Как результат, они корректируют свои планы по мере необходимости.

На проекты необходимо смотреть как на быстрое и стабильное генерирование потока новых полезных возможностей и новых знаний, а не как на выполнение ряда последовательных этапов. Проект генерирует два вида новых знаний: знания о продукте и знания о проекте. Каждый из них полезен для уточнения плана разработки продукта и создания наибольшей стоимости для организации.

Agile-команды участвуют в планировании на трех уровнях: планирование релиза, планирование итерации и дневное планирование. Планирование релиза охватывает срок создания релиза — обычно от трех до шести месяцев. Планирование итерации охватывает срок только одной итерации — обычно от двух до четырех недель. Дневное планирование — это результат обязательств членов команды, принимаемых друг перед другом на ежедневных летучках.

Понимание условий удовлетворенности владельца продукта критически важно как для планирования релиза, так и для планирования итерации. При планировании релиза вся команда занимается выработкой подхода к выполнению условий удовлетворенности для релиза, которые включают в себя объем, календарный график и ресурсы. Для успешной выработки такого подхода владельцу продукта иногда приходится смягчать одно или несколько условий удовлетворенности. Аналогичный процесс происходит и при планировании итерации, где условия удовлетворенности включают в себя новые функции, подлежащие реализации, и высокоуровневое тестирование, демонстрирующее правильность реализации этих функций.

Вопросы для обсуждения

1. Как изменилась бы работа над текущим или предыдущим проектом, если бы ваша команда действовала как единое целое?

2. Какие условия удовлетворенности вы идентифицировали для своего текущего проекта? Все ли заинтересованные в проекте лица и участники согласны с ними? Какие риски возникают при осуществлении проекта, когда согласие достигнуто не по всем условиям удовлетворенности?
3. Почему бюджет и календарный график, определенные на рис. 3.2 как условия удовлетворенности, должны рассматриваться при планировании релиза, а не при планировании итерации?

Часть II

Оценка размера

Agile-команды разделяют оценку размера и оценку срока. Для демонстрации разницы приведу такой пример: представьте, что мне нужно переместить большую кучу земли из одного угла двора в другой. Я могу посмотреть на эту кучу, прикинуть возможности своих инструментов (лопаты и тачки) и прямо сказать, что работа займет пять часов. При такой оценке я не делаю расчетов размера, а сразу перехожу к определению срока.

Допустим теперь, что я пошел другим путем: посмотрел на кучу и оценил ее величину. На основе ее размеров я прихожу к выводу, что объем земли составляет примерно $8,5 \text{ м}^3$. Это моя оценка размера данного проекта. Однако оценка одного лишь размера бесполезна в этой ситуации. Нам нужно узнать, сколько времени займет перемещение земли. Поэтому необходимо преобразовать оценку размера ($8,5 \text{ м}^3$) в срок.

Из ярлыка на моей тачке следует, что ее вместимость составляет $0,17 \text{ м}^3$. Разделив $8,5 \text{ м}^3$ на $0,17 \text{ м}^3$, я определяю, что для перемещения земли мне потребуется 50 ездов. По моим прикидкам, для погрузки земли на тачку уйдет 3 минуты, для перемещения тачки на другой конец двора и выгрузки земли — 2 минуты, а на возврат с пустой тачкой к куче — 1 минута. Всего одна езда займет 6 минут. Поскольку мне нужно сделать 50 ездов, расчетный срок работы составит 300 минут, или 5 часов.

Процесс оценки срока для проекта по разработке программного обеспечения показан на рис. II.1.

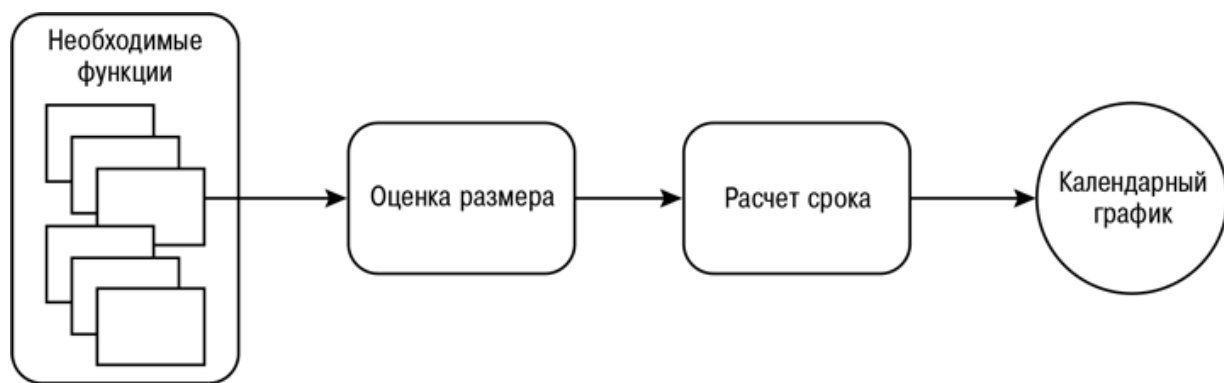


Рис. II.1. Оценка срока выполнения проекта начинается с оценки его размера

В этой части мы сначала рассмотрим два показателя размера: пункты и идеальные дни. Затем мы перейдем к методам оценки и рекомендациям относительно того, когда следует проводить переоценку. Часть II завершается соображениями по выбору предпочтительного показателя размера.

Глава 4

Оценка размера в пунктах

Я могу влезть в туфли даже шестого размера, седьмой мне подходит больше, но покупаю восьмой.

Актриса Долли Партон в кинофильме «Стальные магнолии»

Предположим, у вас под боком открылся новый ресторан и вы решили пойти туда на разведку. На первое вы можете заказать чашку или большую тарелку супа, на второе — полпорции или полную порцию основного блюда, а на третье — маленькую или большую порцию газировки. Скорее всего, вы уже много раз бывали в ресторанах вроде этого и можете без ошибки заказать нужное количество пищи, не спрашивая, сколько грамм супа содержится в маленькой и большой порции, и не уточняя объем основного блюда. Самое большее, о чем вы можете поинтересоваться у официанта, так это о размере салата. Официант, скорее всего, разведет руки, чтобы показать его размер. В таких ситуациях вы делаете заказ, опираясь на относительный, а не на абсолютный размер. Вы говорите: «Принесите мне большую порцию» или «Я хочу маленькую порцию». Никто не называет точный размер, например «Я хотел бы 400 г газировки, 170 г лазаньи и 90 г хлеба».

Аналогичным образом можно оценивать пользовательские истории или функции в agile-проекте. Заказывая большую порцию газировки в незнакомом мне ресторане, я в реальности не знаю, сколько именно граммов мне подадут. Мне известно лишь то, что большая порция газировки больше маленькой и меньше двойной порции. Я также знаю из опыта, что, когда мне хочется пить как сейчас, большая порция газировки, подаваемая в других ресторанах, будет тем, что нужно. К счастью, этого знания мне вполне достаточно. А в проектах по разработке программного обеспечения ситуация еще проще: все, что нужно знать, это то, больше или меньше конкретная история или функция по сравнению с другими историями и функциями.

Пункты — относительный показатель

Пункты — это единица измерения общего размера пользовательской истории, функции или другой части работы. При использовании этой единицы измерения размера мы присваиваем определенное количество пунктов каждому элементу. Величина исходных размеров не играет никакой роли. Что важно, так это относительные размеры. История, которой присвоено два пункта, должна быть в два раза больше, чем история, которой присвоен один пункт. Она должна также составлять по размеру две трети истории, которую оценивают в три пункта.

Количество пунктов, присвоенных истории, представляет ее общий размер. Какой-то определенной формулы определения размера истории не существует. Размер в пунктах — это, скорее, сочетание трудоемкости разработки функции, сложности ее разработки, риска, присущего разработке, и т.п.

Существует два общепринятых подхода к оценке размера в пунктах. Первый заключается в выборе наименьшей истории среди тех, с которыми вы будете работать, и присвоении ей размера, равного одному пункту. Второй предполагает выбор истории среднего размера и присвоение ей значения из середины того диапазона пунктов, который вы будете использовать. Я лично предпочитаю использовать диапазон от 1 до 10. (Причина этого объясняется в главе 6 «Методы оценки».) Иначе говоря, я выбираю историю среднего размера и присваиваю ей значение 5 пунктов. После того как вы произвольным образом установили количество пунктов для первой истории, каждая последующая история оценивается относительно первой или любой другой уже оцененной истории.

Чтобы понять, как это работает, лучше всего попробовать данный метод на практике. Для наглядности используем не пользовательские истории, а породы собак. Будем считать, что пункт в этом случае представляет высоту собаки в холке. Итак, оценим в пунктах следующие породы:

- лабрадор-ретривер;
- терьер;
- немецкий дог;
- пудель;
- такса;
- немецкая овчарка;
- сенбернар;
- бульдог.

Прежде чем читать дальше, прикиньте, сколько пунктов вы бы присвоили каждой породе. Если проделать это, то наши последующие

рассуждения станут понятнее.

Мои оценки приведены в табл. 4.1. Я присваивал эти значения, начиная с лабрадора-ретривера. Размер собак этой породы, на мой взгляд, можно отнести к среднему, поэтому я присвоил им значение 5. Немецкий дог будет, пожалуй, в два раза выше, поэтому он получает 10. Сенбернар значительно выше лабрадора, но чуть ниже дога — он получает 9. Такса по росту самая маленькая и больше чем на 1 не тянет. Бульдог повыше, но тоже не вышел ростом, поэтому я даю ему 3. Если бы я присваивал пункты по весу собак, то бульдог получил бы больше.

Таблица 4.1. Одно из возможных распределений пунктов при оценке собак

Порода собак	Пункты
Лабрадор-ретривер	5
Терьер	3
Немецкий дог	10
Пудель	3
Такса	1
Немецкая овчарка	5
Сенбернар	9
Бульдог	3

Agile-проект нередко начинается с итерации с неполным набором требований, детали которых выясняются в процессе работы. Так или иначе, нам нужно дать оценку всем историям, в том числе и тем, которые сформулированы нечетко. Вы уже видели, как это делается на примере присвоения пунктов пуделю и терьеру. Без дополнительной информации оценить их было бы трудно. Как известно, есть карликовые, средние и стандартные пудели и все они разного размера. Аналогичным образом терьеры — это группа, включающая в себя более двух десятков пород. Одни терьеры (белый высокогорный терьер, норвич-терьер, норфолкский терьер) имеют в холке менее 30 см, а другие (эрдельтерьер) — почти 60 см.

Когда вам дают свободно изложенную пользовательскую историю (или описание собаки), вы делаете определенные допущения, строите

предположения и принимаете решение. В табл. 4.1 я выдвинул предположения в отношении терьера и пуделя и присвоил им по 3 пункта. По моему разумению, даже самые крупные из них меньше лабрадора-ретривера, а самые мелкие не заслуживают больше 1–2 пунктов. Поэтому среднее значение, равное 3, выглядит довольно правдоподобным.

Скорость

Чтобы понять, в чем польза оценки в безразмерных пунктах, введем новое понятие — «скорость». *Скорость* — это показатель темпа продвижения команды. Для ее определения суммируют количество пунктов, присвоенных каждой пользовательской истории, которую команда реализовала в течение итерации[3]. Если команда реализует три истории, каждая из которых оценивается в 5 пунктов, то ее скорость равна 15, а если две истории по 5 пунктов — 10.

Если команда выполнила работу объемом 10 пунктов в течение предыдущей итерации, то мы можем предположить, что она выполнит работу объемом 10 пунктов и в течение текущей итерации. Поскольку пункты характеризуют относительный размер, такое предположение будет справедливым, если команда реализует и две истории по 5 пунктов, и пять историй по 2 пункта.

Во введении к настоящей части этой книги модель, представленная на рис. 4.1, использовалась для демонстрации того, как оценку размера превратить в оценку срока и календарный график. Здесь мы ее приводим, чтобы показать, как она сочетается с пунктами и скоростью.

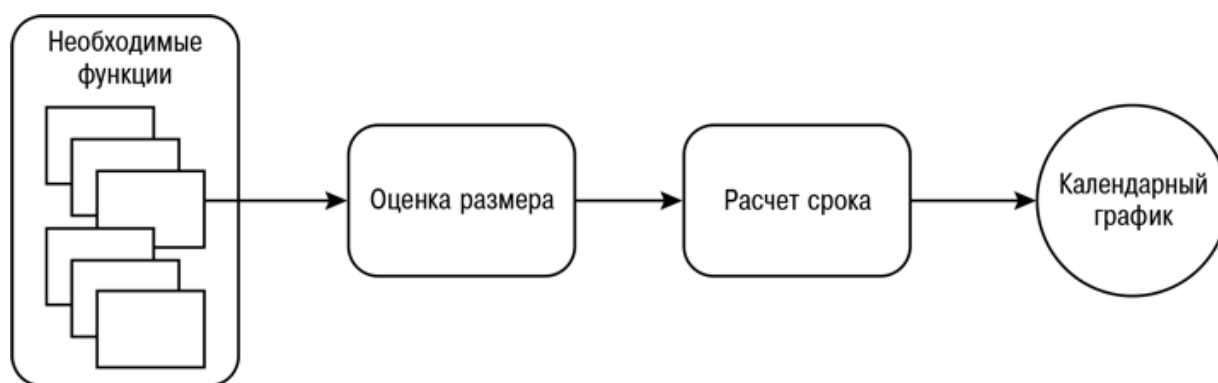


Рис. 4.1. Оценка срока выполнения проекта начинается с оценки его размера

Если суммировать оценки в пунктах для всех необходимых функций, то мы получим совокупный размер проекта. Если известна скорость работы

команды, то можно разделить размер на скорость и получить расчетное количество итераций. Этот показатель срока можно превратить в календарный график, отобразив итерации в определенном порядке в календаре.

Ключевой момент agile-подхода к оценке и планированию заключается в том, что мы оцениваем размер и определяем на его основе срок. Предположим, что мы оценили все пользовательские истории и сумма этих оценок составила 100 пунктов. Это расчетный размер системы. Допустим, из прошлого опыта нам известно, что скорость команды составляет 10 пунктов за двухнедельную итерацию и что команда обеспечит такую же скорость в данном проекте. Зная расчетный размер и нашу скорость, мы можем определить срок выполнения 10 итераций — 20 недель. Отсчитаем 20 недель в календаре и получим календарный график.

Это предельно упрощенное объяснение процедуры планирования релиза. Мы рассмотрим ее более подробно в части IV «Составление календарных графиков». Кроме того, данный пример такой простой по той причине, что мы использовали прошлую скорость команды. Это не всегда возможно — иногда приходится оценивать саму скорость. Определение скорости может оказаться сложным делом, однако существуют определенные методы расчета, помогающие найти ее. Мы рассмотрим их в главе 16 «Оценка скорости».

Скорость обеспечивает корректировку ошибок оценки

К счастью, как только команда начинает продвигаться в реализации проекта, первые несколько итераций наглядно демонстрируют ее скорость. Прелесть использования пунктов для оценки размера заключается в том, что ошибки планирования автоматически корректируются в результате применения показателя скорости. Предположим, что команда оценивает размер проекта в 200 пунктов. Первоначально она исходит из того, что сможет выполнять 25 пунктов за итерацию и, таким образом, завершит проект за восемь итераций. Однако с началом работ выясняется, что фактическая скорость составляет всего 20 пунктов. Без переоценки каких-либо работ команда безошибочно определяет, что для выполнения проекта потребуются 10 итераций, а не восемь.

Чтобы понять, как это работает, представьте, что вас подрядили покрасить дом, который вы никогда не видели. Вам показывают план помещений (рис. 4.2) и говорят, что обеспечат всеми необходимыми материалами, включая кисть, валик и краску. Вам хотелось бы знать, сколько времени уйдет на эту работу, поэтому вы проводите предварительную оценку. Поскольку все, что у вас есть, — это план помещений и вы еще не видели самого дома, оценка основывается на

информации, которую можно получить из плана. Допустим, что трудозатраты на покраску каждой из небольших спален вы оцениваете в 5 пунктов. Цифра «пять» не несет в себе никакого физического смысла, однако она показывает, что трудозатраты на покраску каждой спальни будут одинаковыми. Хозяйская спальня примерно в два раза больше других спален, поэтому вы оцениваете трудозатраты на нее в 10 пунктов.

Теперь посмотрите еще раз на рис. 4.2 и обратите внимание на то, что на плане нет размеров. Какие габариты у двух спален — $2,5 \times 3,0$ м или $4,8 \times 6,0$ м? По плану это определить невозможно. Можно ли утверждать, что оценки, которые вы дали, совершенно бесполезны на данный момент? Нет. На деле эти оценки по-прежнему полезны, поскольку они представляют собой относительные трудозатраты на покраску каждой комнаты. Если окажется, что спальни в два раза больше, чем вы думали, то хозяйская спальня будет также в два раза больше. Оценки останутся теми же, но из-за того, что площадь помещений оказалась в четыре раза больше, чем вы ожидали, темп выполнения работы будет ниже.

Сильная сторона такой оценки заключается в том, что пункты полностью отделяют *оценку трудозатрат* от *оценки срока*. Конечно, трудозатраты и календарный график взаимосвязаны, однако их разделение позволяет проводить оценку того и другого независимо. Фактически вы уже не оцениваете срок выполнения проекта, а вычисляете его или определяете расчетным путем. Это различие довольно тонкое, но очень важное.

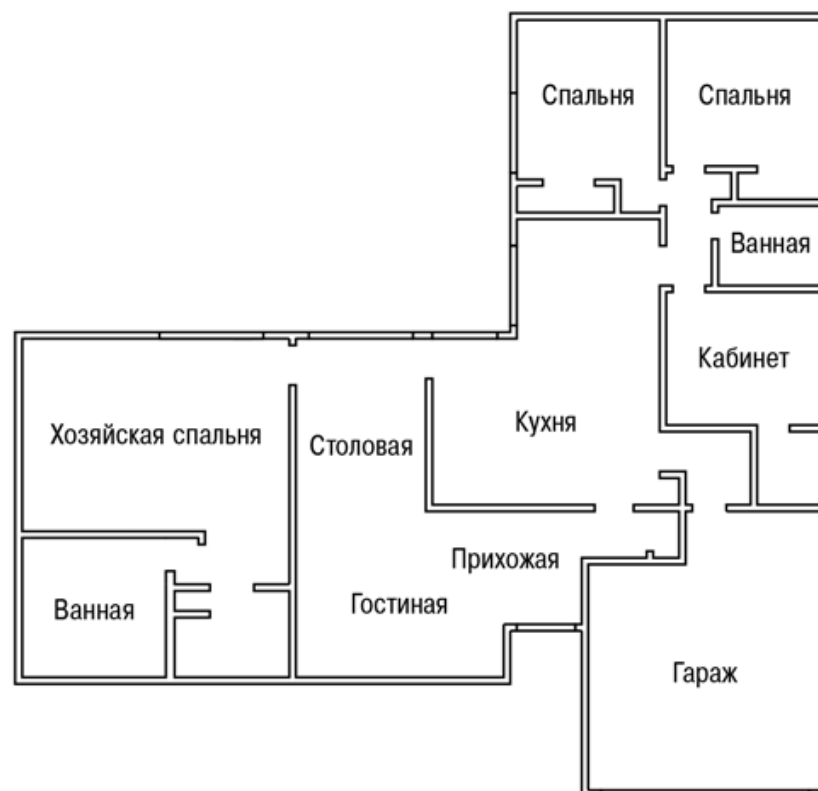


Рис. 4.2. Сколько времени займет покраска этого дома?

Резюме

Пункты — это относительный показатель размера пользовательской истории. Пользовательская история, оцененная в 10 пунктов, в два раза больше, сложнее или рискованнее, чем история, оцененная в 5 пунктов. Аналогичным образом 10-пунктовая история в два раза меньше по размеру, сложности или рискованности, чем 20-пунктовая история. Что здесь по-настоящему важно, так это относительные значения, присвоенные разным историям.

Скорость — это показатель темпа продвижения команды при осуществлении итерации. В конце каждой итерации команда может взять реализованные истории и определить свою скорость путем суммирования оценок всех этих историй в пунктах.

Пункты — это просто оценка размера работы, подлежащей выполнению. Срок проекта определяется не присвоением оценки, а путем деления суммарного количества пунктов на скорость команды.

Вопросы для обсуждения

1. Во сколько пунктов вы бы оценили те или иные функции своего текущего проекта?
2. После оценки собак в пунктах в этой главе какую оценку вы бы присвоили слону, если бы я как клиент вашего проекта сказал, что оговорился и на самом деле хотел предложить вам список млекопитающих, а не собак?
3. Сказать, что две вещи очень близкого размера одинаковы, нетрудно. В пределах какого диапазона размеров (от наименьшего объекта до наибольшего) вы могли бы дать надежную оценку? От одного до пяти? От одного до 10? От одного до 100? От одного до 1000?

Глава 5

Оценка размера в идеальных днях

Для великих свершений нужны две вещи: план и нехватка времени.

Леонард Бернстайн

Сколько времени идет матч в американском футболе?

Вы можете сказать, что в матче четыре 15-минутных периода, поэтому он длится 60 минут. Как вариант, вы можете сказать, что он продолжается порядка трех часов. Каждый ответ в определенном смысле правилен, а их несовпадение показывает разницу между идеальным и общим затраченным временем. *Идеальное время* — это количество времени, которое требуется на что-либо без учета сопутствующих действий. В этом смысле футбольный матч занимает 60 минут. *Общее затраченное время* — это количество времени, которое истекает на часах (или, возможно, на календаре). Общее затраченное на футбольный матч время составляет примерно три часа.

Каждый способ измерения продолжительности имеет свое предназначение. В футбольном матче идеальное время используется судьями для определения момента окончания периодов и матча. Это определенно необходимо. Если же вы собираетесь на стадион посмотреть игру, а ваша супруга спрашивает, как долго это продлится, вряд ли стоит говорить о 60 минутах. Фактически вы будете отсутствовать три часа (плюс время на дорогу туда и обратно).

Практически всегда предсказать продолжительность того или иного события в идеальном времени намного легче, чем указать общее затраченное время. Допустим, вас просят оценить продолжительность конкретного футбольного матча в предстоящие выходные. Если вы решите использовать идеальное время, то можете сразу сказать, что он займет 60 минут. Если же вы захотите дать более точную оценку, то можете взять несколько прошлогодних матчей с добавочным временем, провести кое-какие расчеты и сказать, что, учитывая среднюю историческую продолжительность, в эти выходные матч займет 62,3 минуты.

Теперь предположим, вы решили использовать для оценки общее затраченное время. Одни матчи в прошлом году продолжались два с

половиной часа, а другие больше четырех часов. Разница в определенной мере связана со случайными событиями, такими как травмы, однако она может объясняться стилем игры команд, количеством штрафных санкций и т.п. Телевизионная трансляция матчей длится дольше из-за дополнительного времени, занимаемого рекламой. Будет ли матч более продолжительным или более коротким, зависит от участвующих в нем команд. Чтобы получить такую же точность оценки, как и в случае идеального времени, необходимо учесть все эти факторы.

Конечно, из телевизионной программы я могу узнать, что игра начинается в 13:00 и заканчивается в 16:00. Из этого следует, что кто-то в телевизионной компании оценил общее затраченное время. Что известно компании, а нам нет? Ей известен целый ряд моментов. Во-первых, она собирается добавлять или удалять рекламу в зависимости от того, как быстро будет продвигаться игра. Во-вторых, продолжительность большинства матчей близка к трем часам. Если матч заканчивается раньше, то телевизионная компания добавляет в трансляцию рекламу, интервью или другие заставки. Если матч продолжается дольше, то в чем проблема? Зрители, которые интересуются игрой, будут смотреть ее хоть на 15, хоть на 30 минут дольше. Они не выключат телевизор просто потому, что в программе значится время окончания матча 16:00. Зрители, которым матч не интересен, но которые включили канал, чтобы посмотреть другую передачу в 16:00, тоже, скорее всего, подождут, если задержка не будет слишком долгой. Наконец, за десятилетия трансляции футбольных матчей телевизионная компания приучила нас не рассчитывать на окончание матча минута в минуту.

Этот последний момент очень важен. В результате многолетней практики большинство футбольных болельщиков знают, что время окончания матча 16:00 — всего лишь оценка. Никто не удивится, если игра продлится до 16:15 (превышение на 8%). Болельщики могут расстроиться или рассердиться, но не удивиться. Почему же тогда мы удивляемся, если проект по разработке программного обеспечения, оцененный в 12 месяцев, занимает 13 месяцев (превышение на те же 8%)? Какой срок труднее оценить?

Идеальное время и разработка программного обеспечения

В софтверном проекте идеальное время отличается от общего затраченного времени не из-за перерывов, неудачных пасов и травм, а из-за естественных непроизводительных издержек, которые мы несем ежедневно. В любой

отдельно взятый день в дополнение к запланированной работе над проектом член команды может отвечать на электронные письма, консультировать поставщика, проводить собеседование с кандидатом на замещение вакантной должности аналитика и присутствовать на паре совещаний. Дополнительные примеры причин, по которым идеальное время не совпадает с общим затраченным временем:

- поддержка текущего релиза;
- отсутствие из-за болезни;
- совещания;
- демонстрации продукта;
- работа с персоналом;
- телефонные переговоры;
- специальные проекты;
- повышение квалификации;
- электронная переписка;
- анализ сделанного и прогоны;
- собеседования с кандидатами;
- переключение на другие задачи;
- устранение ошибок в текущих релизах;
- управленческий анализ.

Кроме того, при выяснении причин, по которым идеальное время не соответствует общему затраченному, учтите, что руководители могут непрерывно работать от одного отвлекающего события до другого не более пяти минут (Hobbs, 1987). Даже если типичный разработчик отвлекается в три раза реже, время его непрерывной работы составляет всего 15 минут.

Проблемы могут возникать, когда руководитель задает члену команды неизбежный вопрос: «Сколько времени на это потребуется?» Член команды отвечает «Пять дней», а руководитель отсчитывает пять дней в своем календаре и помечает срок окончания работы жирным красным знаком X. Член команды, однако, на самом деле хотел сказать: «Пять дней, если я буду заниматься только этим, но у меня полно других дел, поэтому примерно две недели».

В софтверном проекте многозадачность еще больше увеличивает разрыв между идеальным и общим затраченным временем. Тренер никогда не скажет футболисту: «Поскольку ты занят не в каждой игре, сыграй-ка еще в этом очень важном хоккейном матче». Разработчик программного обеспечения, которого заставляют работать в многозадачном режиме, в значительной мере теряет свою эффективность в результате переключения между двумя (или более) задачами.

В софтверном проекте мы можем оценивать пользовательские истории или другую работу в *идеальных днях*. При оценке в идеальных днях следует

исходить из следующего:

- оцениваемая история — это единственная задача, над которой вы работаете;
- все, что вам необходимо, есть под рукой в момент начала работы;
- перерывы и отвлечения отсутствуют.

Идеальные дни как показатель размера

При оценке числа идеальных дней, необходимых для реализации той или иной пользовательской истории, не нужно учитывать влияние непроизводительных издержек, обусловленных средой, в которой работает команда. Если на разработку конкретной экранной формы мне необходим один идеальный день, то я буду заниматься ею один идеальный день, независимо от того, работаю ли я в стартапе без непроизводительных издержек, в условиях, где меня отвлекают другие сотрудники, или в жесткой бюрократической системе. Количество времени, которое пройдет по часам (или по календарю), конечно, будет другим. Возможно, мне удастся затратить на работу немногим больше идеального дня в стартапе с низкими непроизводительными издержками. Чем больше будет отвлекающих моментов, тем меньше времени у меня останется на разработку продукта для проекта и тем больше будет срок выполнения работы, рассчитанной на один идеальный день.

Если оставить в стороне непроизводительные издержки организации, то идеальные дни можно использовать для оценки размера точно так же, как и пункты. Затем оценку размера в идеальных днях можно преобразовать в расчетный срок выполнения работы с помощью скорости аналогично тому, как это делается с оценкой размера в пунктах.

Одна оценка, а не множество

Если вы выбираете идеальные дни для оценки, присваивайте одно общее значение каждой пользовательской истории. Некоторые команды пытаются оценивать количество идеальных дней для каждого работника или группы, которые работают с пользовательской историей. Например, команда может сделать такую разбивку для конкретной пользовательской истории: два идеальных дня — программист, один идеальный день — инженер по базам данных, один идеальный день — дизайнер пользовательского интерфейса и два идеальных дня — тестировщик. Мне встречались команды, которые записывают оценки по ролям на карточке истории разноцветными

маркерами или прикрепляют к ней разноцветные стикеры.

Обычно я не советую этого делать. На данном этапе концентрация внимания на индивидуальных ролях в команде отвлекает от идеологии «мы все работаем над этим вместе», которую хотелось бы видеть в agile-команде. Помимо прочего, это очень сильно увеличивает объем работы при планировании релиза. Если в каждой истории делать оценку для каждой роли, задействованной в работе, план релиза должен корректно учитывать эти роли, а в таком случае необходимо отслеживать скорость и оставшийся объем работы для каждой роли.

Хотя на это редко когда стоит тратить силы, иногда такой подход просто необходим. Не так давно у меня был клиент, работающий с тремя версиями продукта — одна для Macintosh, другая для Windows и третья для карманных компьютеров. В этом случае критически важно обеспечить абсолютно одинаковую функциональность. Более того, члены команды на тот момент не могли переключаться с разработок для Mac на разработки для Windows и карманных компьютеров. В такой ситуации оценка идеального времени для каждой роли по каждой истории может быть полезна. Следует, однако, учитывать, что это увеличивает бремя администрирования.

Резюме

Идеальное время и общее затраченное время — не одно и то же. Идеальное время игры в американский футбол составляет 60 минут (четыре 15-минутных периода). Вместе с тем для завершения 60-минутного матча обычно требуется около трех часов. Такая разница объясняется наличием перерывов в игре.

Количество времени, необходимое для реализации пользовательской истории, легче оценивать в идеальных днях, а не в затраченных днях. Оценка в затраченных днях требует учета всех перерывов, которые могут возникнуть в процессе работы над историей. Если же для оценки используются идеальные дни, то учитывается только время реализации истории. Таким образом, идеальные дни характеризуют размер работы, хотя и не так строго, как пункты.

При использовании идеальных дней лучше всего давать одну общую оценку каждой пользовательской истории. Вместо разбивки оценки пользовательской истории по ролям (четыре идеальных дня — программист, два идеальных дня — тестировщик и три идеальных дня — владелец продукта) лучше дать суммарную оценку и сказать, что работа над историей в целом займет девять идеальных дней.

Вопросы для обсуждения

1. Если идеальный день — это восемь часов непрерывной, целенаправленной работы, то какое общее затраченное время в часах соответствует одному идеальному дню в ваших условиях?
2. Можно ли как-то улучшить ситуацию?

Глава 6

Методы оценки

Давать предсказания очень трудно, особенно когда они касаются будущего.

Нильс Бор, датский физик

Чем больше сил и средств мы вкладываем во что-то, тем лучше результат. Верно? Возможно, но нередко для получения *приемлемых* результатов требуется лишь часть этих затрат. Например, моя машина грязная и ее нужно помыть. Если я займусь этим сам, то потрачу на мойку около часа — этого достаточно, чтобы вымыть машину снаружи, пропылесосить салон и протереть окна. Затратив один час, я получу сравнительно чистый автомобиль.

В качестве альтернативы можно воспользоваться услугой по чистке автомобиля. Там на эту процедуру уйдет четыре часа — мастера проделают все то же, что и я, но несравненно более тщательно. Они также обработают кузов полиролью, отполируют переднюю панель и т.д. Однажды я наблюдал, как они чистят ватными палочками труднодоступные места, куда с тряпкой не добраться. При этом масса сил и энергии тратится на получение чуть лучших результатов. Когда я сам занимаюсь этим, закон уменьшения отдачи заставляет меня остановиться задолго до того, как дело дойдет до ватных палочек.

Об уменьшении отдачи на затраченное время необходимо помнить и при оценке. Зачастую мы, почти не раздумывая над оценкой, называем определенное число, которое практически ничем не хуже значения, выработанного в результате долгих размышлений. Взаимосвязь между точностью оценки и затраченными усилиями показана на рис. 6.1. Кривая на этом рисунке построена на основании моего опыта, подтвержденного в ходе дискуссий с коллегами. Она не является результатом экспериментальных измерений.



Рис. 6.1. Дополнительные усилия, затраченные на оценку, дают очень незначительный вклад за пределами определенной точки

Для лучшего понимания этой взаимосвязи предположим, что вы решили оценить, сколько печенья я съел за прошлый год. Вы можете без особых усилий просто назвать какое-нибудь число наугад. Отметим его на рис. 6.1 — ваша оценка окажется очень близкой к левому краю оси усилий и вряд ли будет точной. Можно пойти вправо по оси усилий и потратить хотя бы полчаса на поиски данных о среднем потреблении печенья по стране. Это повысит точность по сравнению с оценкой наугад. Если нужна еще более высокая точность, можно провести исследовательскую работу — обзвонить моих друзей и родственников, запросить мои прошлые заказы на печенье в ближайшем магазине и т.д. Можно также последить за мной в течение дня, а лучше месяца, а потом экстраполировать полученные результаты по съеденному печенью на годовой период.

Увязывайте усилия, которые затрачиваются на оценку, с преследуемой целью. Если вы пытаетесь решить, послать мне коробку печенья в подарок или нет, очень точная оценка ни к чему. Если оценка нужна для принятия решения о том, что лучше — разработать программу или купить готовую, чаще всего достаточно знать, что проект займет от шести до 12 месяцев. Возможно, эту оценку придется уточнить настолько, чтобы можно было судить, семь или восемь месяцев займет проект.

Посмотрите повнимательнее на рис. 6.1 и обратите внимание на следующие моменты. Во-первых, не важно, сколько затрачивается усилий, — оценка никогда не достигает максимума по оси точности. Не имеет значения, ценой каких затрат получена оценка, — как ни крути, это не более чем оценка. Никакие дополнительные затраты сил и средств не сделают оценку идеальной. Во-вторых, заметьте, как мало усилий требуется для кардинального повышения точности по сравнению с нулевой линией. Из рис. 6.1 следует, что всего 10% усилий обеспечивают достижение 50% от

потенциально возможной точности. Наконец, обратите внимание на то, что в конечном итоге точность оценки падает. Не исключено, что при чрезмерно больших затратах сил и средств на оценку результат будет менее точным.

Перед тем как разрабатывать план проекта, полезно подумать о том, в какой точке кривой на рис. 6.1 мы хотим быть. Во многих проектах в стремлении достичь очень высокого уровня на оси точности команды заходят очень далеко на оси усилий, хотя выгоды от этого быстро уменьшаются. Нередко это результат упрощенческих представлений о том, что можно зафиксировать бюджеты, календарные графики и объемы работ и что успех проекта эквивалентен выполнению проекта в срок и в рамках бюджета с поставкой заранее определенного и точно спланированного набора функций. Такой подход порождает склонность к разработке объемной документации с техническими требованиями, раздуванию объемов предварительного анализа и составлению детальных планов, отражающих все задачи, о которых команда может помыслить. Но даже после всей этой дополнительной предварительной работы оценки не становятся идеальными.

Agile-команды предпочитают держаться ближе к левому краю графика на рис. 6.1. Они признают невозможность устранения неопределенности оценок, однако считают, что небольшие усилия приносят большой результат. Хотя agile-команды не так далеко продвигаются по осям точности/усилий, их планы более надежны, поскольку они часто поставляют полностью работоспособные, протестированные, интегрированные программы с небольшими дополнениями.

Оценки — продукт совместной работы

Оценкой занимается не какой-то один представитель команды. Agile-команды не полагаются на оценку единственного эксперта. Несмотря на хорошо известный факт, что оценка, подготовленная тем, кто выполняет работу, лучше оценки, данной кем-то другим (Lederer and Prasad, 1992), наилучшими являются коллективные оценки членов команды, которые участвуют в работе. Это объясняется двумя причинами.

Во-первых, в agile-проекте обычно неизвестно, кто будет выполнять ту или иную задачу. Конечно, мы можем предполагать, что сложной задачей разработки хранимой процедуры будет заниматься входящий в команду гуру по базам данных. Однако нет никаких гарантий, что это будет именно так. Он может быть занят, когда дело дойдет до разработки, и работу выполнит кто-то другой. Поскольку любой член команды может получить

любую задачу, очень важно, чтобы каждый внес свой вклад в оценку.

Во-вторых, даже если мы ожидаем, что выполнять эту работу будет гуру по базам данных, у других членов команды могут быть свои соображения в отношении его оценки. Допустим, гуру команды по базам данных Кристи оценивает определенную пользовательскую историю в три идеальных дня. Однако другой участник проекта, хотя он и не обладает достаточными знаниями для самостоятельной разработки программы, может сказать: «Кристи, ты сошла с ума — в прошлый раз, когда мы занимались похожей функцией, работа заняла намного больше времени. Мне сдается, что ты забыла, насколько сложной оказалась задача тогда». Кристи, конечно, может объяснить, почему в этот раз все будет по-другому. Мой опыт, впрочем, показывает, что, скорее всего, она согласится с тем, что занизила оценку функции.

Шкала оценки

Исследования показывают, что мы лучше всего оцениваем величины в пределах одного порядка (Miranda, 2001; Saaty, 1996). В своем городе вы должны довольно хорошо оценивать относительные расстояния до таких объектов, как ближайший продовольственный магазин, ближайший ресторан и ближайшая библиотека. Например, библиотека может находиться в два раза дальше ресторана. Результаты будут намного менее точными, если вас попросят оценить относительное расстояние до Луны или до столицы соседнего государства. Поскольку мы лучше всего оцениваем величины в пределах одного порядка, большинство оценок должны укладываться именно в такой диапазон.

Я использую следующие две шкалы оценки:

- 1, 2, 3, 5 и 8;
- 1, 2, 4 и 8.

В основе каждой из этих последовательностей лежит своя логика. Первая — последовательность Фибоначчи[4]. Я считаю эту последовательность очень полезной, потому что шаг в ней повышается соответствующим образом с ростом величины чисел. Шаг в один пункт между числами 1 и 2, а также между числами 2 и 3 выглядит подходящим, как и шаги между 3 и 5 и между 5 и 8. Во второй последовательности каждое число определяется путем умножения предыдущего числа на два. Эти нелинейные последовательности работают хорошо, поскольку отражают повышение неопределенности, связанной с оценками более крупных элементов работы. В принципе, данные последовательности равноценны, но лично я

предпочитаю первую.

Каждое из этих чисел следует рассматривать как емкость, в которую «выливают» объекты соответствующего размера. Работу лучше представлять как текущий песок, а не воду, льющуюся в емкость. Если вы используете ряд 1, 2, 3, 5 и 8, а оцениваемая история чуть больше, чем другие оцененные в пять пунктов истории, то ее можно поместить в 5-пунктовую емкость. Понятное дело, что история, которую вы оцениваете на семь, для 5-пунктовой емкости не подходит.

Вы вполне можете включить ноль в качестве приемлемого числа в свой диапазон оценки. Хотя у команды вряд ли будет много пользовательских историй или функций, которые действительно не требуют трудозатрат, включение нуля нередко полезно. Причин для этого две. Во-первых, если мы хотим уместить все функции в диапазон, не превышающий 10, то присвоение ненулевых значений самым маленьким функциям ограничит размер самых крупных функций. Во-вторых, если работа реально ближе к нулю, чем к единице, то команда может не захотеть, чтобы реализация такой функции учитывалась при определении скорости. Если команда получит один пункт в этой итерации за что-то действительно мелкое, то в следующей итерации она либо потеряет единицу в скорости, либо ей придется зарабатывать этот пункт, выполняя не такую мелкую работу.

Если команда решит не включать ноль в шкалу оценки, то все участники проекта (особенно владелец продукта) должны ясно понимать, что $13 \times 0 \neq 0$. У меня никогда не было проблем с объяснением этого владельцам продукта, которые понимали, что 0-пунктовая история — эквивалент бесплатного сыра. Однако они также понимали, что в одной итерации количество ломтиков бесплатного сыра не может быть безграничным. Альтернативой использованию нуля является группирование очень маленьких историй и их оценка как отдельный элемент работы.

Некоторые команды предпочитают работать с большими числами, такими как 10, 20, 30, 50 и 100. Это нормально, поскольку они представляют диапазон одного порядка. Вместе с тем, если вы имеете дело с большими числами в диапазоне от 10 до 100, я настоятельно рекомендую заранее определить используемые числа в этом диапазоне. Не допускайте, например, оценки одной истории в 66 пунктов или идеальных дней, а другой — в 67. Это ложный уровень точности, так как невозможно уловить 1,5%-ную разницу в размере. Разница в один пункт приемлема для таких величин, как 1, 2 и 3. В процентном выражении она значительно больше разницы между 66 и 67.

Пользовательские истории, эпосеи и сюжеты

Хотя обычно мы стремимся оценивать пользовательские истории, которые

имеют размеры одного порядка, так бывает не всегда. Если оценивать все в пределах одного порядка, это означает, что истории придется описывать на довольно близком уровне. Для функций, в необходимости которых нет уверенности (прежде чем вкладывать в них слишком много, желательно провести предварительную оценку затрат), или функций, которые могут не понадобиться в ближайшем будущем, нередко желательна одна значительно более крупная пользовательская история. Более крупную пользовательскую историю иногда называют *эпопеей*.

Помимо этого, ряд взаимосвязанных пользовательских историй можно объединить (обычно с помощью скрепки, если в работе используются карточки) и работать с ними как с единым объектом в целях оценки или планирования релиза. Такой ряд пользовательских историй называют *темой*. Эпопея в силу своего размера нередко является темой.

Объединяя группы историй в темы и описывая некоторые истории как эпопеи, команда может сократить трудоемкость оценки. Вместе с тем важно понимать, что оценки тем и эпопей будут более неопределенными, чем оценки более конкретных, маленьких пользовательских историй.

Пользовательские истории, подлежащие реализации в ближайшем будущем (в следующих нескольких итерациях), должны иметь не очень большой размер, чтобы работу над ними можно было завершить за одну итерацию. Такие объекты следует оценивать в пределах одного порядка. Я использую для этого последовательность 1, 2, 3, 5 и 8.

Пользовательские истории или другие объекты, работа с которыми, скорее всего, не будет осуществляться в ближайших итерациях, можно представить в виде эпопей или тем. Такие объекты можно оценивать в единицах за пределами рекомендуемого мною диапазона от 1 до 8. Чтобы оценивать крупные объекты, я добавляю 13, 20, 40 и 100 к своей любимой последовательности 1, 2, 3, 5 и 8.

Получение оценки

Наибольшее распространение получили следующие три метода оценки:

- экспертная оценка;
- оценка по аналогии;
- разбивка на более мелкие части.

Каждый из этих методов может применяться индивидуально, однако для получения наилучших результатов их следует комбинировать.

Экспертная оценка

Если вы хотите знать, сколько времени займет то или иное дело, спросите эксперта. Именно таков один из подходов. При оценке на основе экспертных заключений эксперта спрашивают, сколько времени потребуется на что-то или насколько большим это окажется. Эксперт дает оценку, опираясь на свою интуицию или чутье.

Данный подход больше пригоден для традиционных проектов, а не для agile-проектов. В agile-проекте оценки присваиваются пользовательским историям или другой необходимой для пользователей функциональности. Разработка такой функциональности обычно требует участия нескольких специалистов с разной квалификацией. Это усложняет поиск подходящих экспертов, поскольку они должны уметь оценивать трудозатраты по всем квалификациям. В традиционном проекте, где оценки связаны с задачами, эта проблема не так остра в силу того, что каждую задачу обычно выполняет один человек.

Большим плюсом экспертной оценки является то, что она, как правило, не занимает много времени. Обычно разработчик читает пользовательскую историю, задает пару-тройку уточняющих вопросов и дает оценку, опираясь на свою интуицию. По некоторым данным, такой вид оценки даже более точен, чем другие, более аналитические подходы (Johnson et al., 2000).

Оценка по аналогии

Альтернативой экспертной оценке является оценка по аналогии. Именно к ней мы прибегаем, когда говорим: «Эта история немного больше, чем та история». При оценке по аналогии оценщик сравнивает оцениваемую историю с одной или несколькими другими историями. Если данная история превышает другие по размеру в два раза, то говорят, что она в два раза больше. Существуют явные свидетельства того, что мы оцениваем относительные размеры лучше, чем абсолютные (Lederer and Prasad, 1998; Vicinanza et al., 1991).

При использовании этого метода никто не сравнивает все истории с общей базой или универсальным эталоном. Каждую новую историю оценивают относительно разных, уже оцененных историй. Такой подход называют триангуляцией. Суть его в том, что оцениваемая история сравнивается с двумя другими. Историю оценивают в пять пунктов, если она выглядит немного больше истории, оцененной в три пункта, но немного меньше истории, оцененной вами в восемь пунктов.

Разбивка на более мелкие части

Разбивка предполагает разделение истории или функции на более мелкие, легче поддающиеся оценке части. Если большинство пользовательских

историй, включаемых в проект, требуют для реализации от двух до пяти дней, крайне трудно оценить историю, для реализации которой нужно, скажем, 100 дней. Дело не только в том, что крупные объекты труднее поддаются оценке, в этом случае просто очень мало сходных историй для оценки. Спросить «действительно ли эта история в 50 раз сложнее, чем та?» — совершенно иное дело, чем спросить «действительно ли на эту историю нужно в полтора раза больше времени, чем на ту?».

Для решения подобной проблемы нужно разбить большую историю или функцию на несколько более мелких объектов, а затем провести их оценку. Вместе с тем применять такой подход следует осмотрительно, чтобы не зайти слишком далеко. Наиболее наглядно эту проблему демонстрирует следующий, не относящийся к программному обеспечению пример. Воспользуемся методом разбивки для оценки моего счета в гольфе в эти выходные. Будем считать, что на поле, где я играю, 18 лунок, каждая из которых классифицируется как «пар 4». (Если вы не знакомы с правилами подсчета очков в гольфе, то замечу, что пар — это количество ударов, которые должен сделать игрок, чтобы загнать мяч в лунку.)

Чтобы сделать оценку методом разбивки, нужно оценить мой результат на каждой лунке. Первая лунка довольно легкая, поэтому на ней дадим мне три. На следующей лунке я обычно загоняю мяч в озеро, поэтому здесь семь. Еще там есть лунка с преградой «сэндтрэп»; пусть здесь будет пять. И так далее. Однако если я мысленно представлю все поле, то, скорее всего, забуду об одной из лунок. Конечно, обнаружить это очень легко, поскольку, как известно, должно быть 18 индивидуальных оценок. Когда же мы разбиваем историю, такого контрольного теста у нас нет.

Чрезмерная разбивка приводит не только к повышению вероятности пропуска задачи, суммирование оценок по множеству мелких задач также ведет к проблемам. Например, для каждой из 18 лунок я могу оценить свой результат в диапазоне от 3 до 8. Умножение оценок на 18 дает мне полный диапазон от 54 до 144. Вряд ли я пройду все лунки так хорошо или так плохо. Если меня попросят оценить общий результат, то я, скорее всего, назову диапазон от 80 до 120, что меньше полного диапазона и ближе к реальности.

Конкретная рекомендация по разбивке пользовательских историй приведена в главе 12 «Разбивка пользовательских историй».

Покер планирования

На мой взгляд, наилучший для agile-команд способ оценки — это покер планирования (Grenning, 2002). Покер планирования объединяет

экспертную оценку, оценку по аналогии и разбивку в удобный подход, позволяющий быстро получить надежные результаты.

В покере планирования участвуют все разработчики команды. Напомню, что под *разработчиками* понимаются программисты, тестировщики, администраторы баз данных, аналитики, дизайнеры пользовательских интерфейсов и т.д. В agile-проекте их обычно не более десятка. Если разработчиков больше, то их лучше разделить на две команды. Каждая команда может проводить независимую оценку, что позволяет уменьшить размер проекта. Владелец продукта тоже участвует в покере планирования, но оценкой не занимается.

Перед началом процесса каждому оценщику дают колоду карт. На каждой карте написана одна из возможных оценок. Оценщик может получить, например, колоду карт, на которых написано 0, 1, 2, 3, 5, 8, 13, 20, 40 и 100. Карты необходимо подготовить до начала игры в покер планирования, а цифры, написанные на них, должны быть достаточно крупными, чтобы разглядеть их с другого конца стола. Карты можно сохранить и использовать в следующей сессии покера планирования.

Ведущий зачитывает описание каждой оцениваемой пользовательской истории или темы. Роль ведущего обычно берет на себя владелец продукта или аналитик. Впрочем, ведущим может быть любой участник, поскольку эта роль не предполагает получения каких-либо особых привилегий. Владелец продукта отвечает на все вопросы оценщиков, если они возникают. Всех участников просят не забывать о кривой усилий/точности (рис. 6.1). Цель покера планирования заключается не в получении оценки, которая выдержит все последующие проверки, а в сохранении позиции ближе к левому краю оси усилий, где полезную оценку можно получить при небольших затратах.

После получения ответов на все вопросы каждый оценщик самостоятельно выбирает карту, соответствующую его оценке. Карты не открывают до тех пор, пока все оценщики не определятся со своим выбором. Затем карты одновременно открываются, чтобы все участники могли видеть оценки.

Чаще всего в этот момент оценки различаются очень сильно. На деле это совсем неплохо. Когда оценки различаются, оценщики с максимальным и минимальным результатами объясняют причину своего выбора. Очень важно, чтобы здесь на оценщиков не обрушилась критика. Нам нужно лишь выяснить, что заставило их прийти к такому мнению.

Например, оценщик с максимальным результатом может сказать: «Ну, чтобы протестировать эту историю, нам нужно создать мок-объект. На это может уйти целый день. Кроме того, я не уверен, что стандартный алгоритм сжатия будет эффективен, а раз так, то нам, возможно, придется писать новый». Оценщик с минимальным результатом может сказать: «Я полагал,

что мы будем хранить эту информацию в XML-файле — это было бы легче, чем создавать базу данных. К тому же я не подумал об увеличении объема данных — это может стать проблемой».

Группа может кратко обсудить историю и полученные оценки. Ведущему позволяет делать любые замечания, которые, с его точки зрения, будут полезны при программировании и тестировании данной истории. После обсуждения каждый оценщик проводит переоценку и выбирает новую карту. Здесь карты опять не открывают до тех пор, пока все не сделают выбор, затем участники одновременно показывают свои результаты.

Во многих случаях уже во втором раунде оценки сходятся. Если этого не происходит, повторите процесс. Цель заключается в получении одной оценки, которую можно использовать для истории. Редко когда требуется более трех раундов оценки, однако продолжайте процесс до тех пор, пока оценки не сблизятся. Совершенно не обязательно, чтобы все присутствующие открыли карту с одной и той же оценкой. Если бы я вел заседание, посвященное оценке, и во втором раунде результаты четырех оценщиков выглядели бы как 5, 5, 5 и 3, то я бы спросил оценщика с минимальным результатом, не согласится ли он на оценку 5. Подчеркну, что целью является не абсолютная точность, а разумность.

Правильная продолжительность дискуссии

Предварительное обсуждение дизайна всегда полезно для оценки. Вместе с тем чересчур длительное обсуждение приводит к слишком сильному смещению команды вправо по кривой «усилия/точность» (рис. 6.1). Существует простой способ поддержать дискуссию, но не дать ей затянуться.

Приобретите песочные часы на две минуты и поставьте их на центр стола, за которым сидят участники покера планирования. Любой из присутствующих может в любой момент перевернуть часы. Когда песок кончается (истекают две минуты), начинается следующий раунд игры в карты. Если согласие не достигается, дискуссию можно продолжить, но при этом кто-нибудь должен сразу же перевернуть часы, чтобы ограничить ее двумя минутами. Редко когда часы переворачиваются более двух раз. Такой прием со временем помогает команде научиться быстрее приходить к общей оценке.

Сессии с более узким составом участников

Игру в покер планирования можно проводить с частью команды без привлечения всех ее членов. Такой вариант не идеален, но вполне разумен, особенно когда оценить нужно большое число объектов, что не редкость в начале нового проекта.

Для этого команду разбивают на две или три группы, в каждой из которых должно быть не менее трех оценщиков. Очень важно, чтобы оценки всех групп были согласованными. То, что ваша группа оценивает в три пункта или идеальных дня, должно соответствовать тому, что моя группа оценивает так же. Чтобы добиться этого, все группы должны совместно попрактиковаться в покере планирования на протяжении часа или немного дольше. Пусть они оценят порядка 10–20 историй. Позаботьтесь о том, чтобы у каждой группы был экземпляр этих историй с оценками и чтобы группы использовали их в качестве базы для оценки других историй.

Когда играть в покер планирования

Командам необходимо играть в покер планирования в двух случаях. Во-первых, обычно большое число объектов оценивается перед официальным началом проекта или во время его первых итераций. Оценка первоначального набора пользовательских историй может потребовать проведения двух или трех заседаний продолжительностью от одного до трех часов каждое. Естественно, все зависит от количества оцениваемых объектов, размера команды и умения владельца продукта кратко пояснять требования.

Во-вторых, командам нужно оценивать новые истории, идентифицированные в ходе итерации. Один из способов осуществления этого — проведение очень короткого совещания по оценке в конце каждой итерации. Обычно этого вполне достаточно для оценки любой работы, которая появилась в процессе итерации, и учета новой работы в приоритетах следующей итерации.

Как вариант, Кент Бек предлагает повесить на стене конверт и помещать в него все новые истории. Как только у кого-нибудь выпадает свободная минута, он берет из конверта одну-две истории и оценивает их. Команды обычно устанавливают правило, в соответствии с которым все истории должны оцениваться к концу дня или итерации. Мне нравится идея конверта на стене для неоцененных историй. Вместе с тем хотелось бы, чтобы член команды, у которого появилась свободная минута для оценки, мог найти еще кого-то, готового провести совместную оценку.

Почему покер планирования работает

Теперь, когда я описал суть покера планирования, стоит назвать несколько причин, по которым он работает так хорошо.

Во-первых, покер планирования сводит вместе мнения нескольких экспертов и позволяет получить общую оценку. Поскольку эти эксперты образуют кроссфункциональную команду, представляющую все дисциплины программного проекта, они более компетентны в сфере оценки, чем кто-либо другой. На основе тщательного анализа литературы по оценке программного обеспечения Йоргенсен (Jørgensen, 2004) пришел к выводу, что «оценивать задачи должны те, кто наиболее компетентен в их выполнении».

Во-вторых, в процессе игры в покер планирования возникает живой диалог, и оценщики обращаются к мнению коллег для подтверждения своих оценок. Это повышает точность оценки, особенно когда дело касается объектов со значительным уровнем неопределенности (Nagafors and Brehmer, 1983). Обращение за подтверждением оценок в определенной мере компенсирует недостаток информации (Brenner et al., 1996). Это очень важно в agile-проекте, поскольку оцениваемые пользовательские истории нередко намеренно составляются нечетко.

В-третьих, исследования показывают, что усреднение индивидуальных оценок дает лучшие результаты (Hoest and Wohlin, 1998), как и коллективное обсуждение оценок (Jørgensen and Moløkken, 2002). Коллективное обсуждение является основой покера планирования, а такие обсуждения ведут к усреднению различных индивидуальных оценок.

Наконец, покер планирования работает, потому что это занятие доставляет удовольствие.

Резюме

Более значительные затраты времени и усилий на оценку не обязательно повышают точность результата. Трудозатраты на оценку должны определяться целью этой оценки. Хотя все знают, что наилучшие оценки дают те, кто выполняет работу, в agile-команде исполнитель работы не известен заранее. Таким образом, оценка должна представлять собой коллективную деятельность членов команды.

Оценки должны проводиться по заранее определенной шкале. Функции, с которыми будут работать в ближайшем будущем и которые требуют относительно надежной оценки, должны быть достаточно маленькими, чтобы их можно было оценивать по нелинейной шкале от 1 до 10, такой как 1, 2, 3, 5 и 8 или 1, 2, 4 и 8. Крупные функции, которые, скорее всего, не будут реализованы в ближайших нескольких итерациях, можно оставить крупными и оценивать в таких единицах, как 13, 20, 40 и 100. Некоторые команды добавляют в свою шкалу оценки ноль.

Чтобы получить оценку, мы используем такие подходы, как экспертная оценка, оценка по аналогии и разбивка на более мелкие части. Увлекательным и эффективным способом объединения этих подходов является покер планирования. В покере планирования каждый оценщик получает колоду карт с возможными оценками. Оцениваемую функцию обсуждают, а затем каждый оценщик выбирает карту, представляющую его оценку. Карты открывают одновременно. Оценки обсуждают и повторяют процесс до тех пор, пока оценки не сойдутся.

Вопросы для обсуждения

1. Насколько правильны ваши оценки сегодня? Какой метод вы в основном используете: экспертную оценку, оценку по аналогии или разбивку?
2. Какую шкалу оценки вы предпочитаете использовать? Почему?
3. Кто должен участвовать в покере планирования для вашего проекта?

Глава 7

Переоценка

Нет никакого смысла стремиться к точности, когда вы даже не знаете, о чем говорите.

Джон фон Нейман

Один из наиболее часто задаваемых вопросов относительно оценки с использованием пунктов или идеальных дней звучит так: «Когда следует проводить переоценку?» Отвечая на него, важно помнить, что пункты и идеальные дни предназначены для оценки общего размера и сложности реализуемой функции. Так, пункты не характеризуют количество времени, необходимое для реализации функции, хотя мы нередко думаем иначе. Время, которое требуется для этого, является величиной, производной от размера функции (оцененного либо в идеальных днях, либо в пунктах) и темпа продвижения команды (представленного в виде скорости).

Поскольку пункты и идеальные дни характеризуют размер, понятно, что переоценка необходима только при изменении относительного размера истории. При работе с пунктами или идеальным временем переоценка не производится только по той причине, что реализация истории заняла больше времени, чем мы предполагали. Лучше всего продемонстрировать это на примерах.

Знакомство с сайтом SwimStats

Далее в этой главе и в некоторых последующих мы будем работать со SwimStats, гипотетическим веб-сайтом для пловцов и их тренеров. SwimStats предлагается как сервис для команд, разделенных по возрастным группам, школьных команд и команд колледжей. Тренеры используют его для отслеживания результатов пловцов, организации тренировок и подготовки к соревнованиям; пловцы — для ознакомления с результатами соревнований, отслеживания своих результатов и контроля динамики с течением времени. Официальные представители, присутствующие на

соревнованиях по плаванию, вводят результаты в систему. Пример экрана сайта SwimStats приведен на рис. 7.1.

Boulder Valley Summer Swim League			Search	League	Dual Meets	Community	News
Savannah Cohn			Lafayette Seals			Gender: F	
Leagues (Email us.)							
League	Team(s)	Season start date	Events Swam	First	Second	Third	
2005 BVSSL	Lafayette Seals	05/15/2005	18	12	4	2	
Click on Team Name, Swimmer's Name or Meet Date to view more information. Meets are sorted with most recent meet on top.							
Indicates Winner							
07/9/2005 Lafayette Seals at Meadow Hills							
Event Number	Age Group	Event	Time	Place	Points		
14	Girls 9-10	100 Free	2:09.48	1	6		
34	Girls 9-10	50 Back	1:06.41	1	6		
64	Girls 9-10	50 Breast	1:13.97	1	6		
07/16/2003 Lafayette Seals host Huntington Beach Blue Dolphins							
Event Number	Age Group	Event	Time	Place	Points		
14	Girls 9-10	100 Free	2:03.48	1	6		
34	Girls 9-10	50 Back	1:04.41	1	6		
64	Girls 9-10	50 Breast	1:13.43	1	6		

Рис. 7.1. Вид одного из экранов на сайте SwimStats

Когда переоценка не требуется

Используя сайт SwimStats в качестве примера, сначала кратко рассмотрим ситуацию, в которой переоценка не требуется. Предположим, что у нас есть истории, представленные в табл. 7.1. В первой итерации реализуются первые две истории. Команду это не устраивает, поскольку она считает, что должна реализовывать в два раза больше пунктов (12 вместо шести) за итерацию. Она считает, что каждая из реализованных историй была в два раза больше или сложнее, чем казалось вначале, и именно поэтому на них потребовалось в два раза больше времени, чем предполагалось. Команда принимает решение удвоить количество пунктов, связанных с каждой историей. В результате скорость становится равной 12 (две истории по шесть пунктов), что более приемлемо для команды.

До начала проекта, однако, команда сочла все четыре истории в табл. 7.1 равными по размеру и сложности и оценила каждую в три пункта. Поскольку команда по-прежнему считает эти истории равноценными, оценки историй 3 и 4 также необходимо удвоить. Команда увеличивает количество получаемых пунктов за реализованные истории, а это, в свою очередь, приводит к удвоению скорости. Вместе с тем из-за того, что она

также удваивает количество оставшейся работы в проекте, ее положение остается таким же, как если бы она оставила оценку равной трем, а скорость — шести.

Таблица 7.1. Истории и оценки для сайта SwimStats

Номер истории	История	Оценка
1	Как тренер я могу вводить имена и гендерно-возрастную информацию по всем пловцам моей команды	3
2	Как тренер я могу устанавливать расписание тренировок	3
3	Как пловец я могу видеть свое время во всех соревнованиях	3
4	Как пловец я могу корректировать свою гендерно-возрастную информацию	3

Скорость — великий уравниватель

Этот пример показывает, что скорость — великий уравниватель. Поскольку оценка каждой функции дается относительно других функций, не имеет значения, правильна ли эта оценка, не очень правильна или совсем неправильна. Что важно, так это согласованность оценок. Нельзя просто бросить игральную кость и принять выпавшее число в качестве оценки функции. Вместе с тем, пока наши оценки согласованны, измерение скорости в первых нескольких итерациях позволяет составить надежный календарный график.

Рассмотрим другой пример. Предположим, что проект состоит из 50 пользовательских историй, каждая из которых оценена в один пункт. Для простоты будем считать, что я — единственное лицо, работающее над этим проектом, и что я реализую одну историю размером один пункт за рабочий день. Таким образом, за двухнедельную итерацию я предполагаю реализовать 10 историй и добиться скорости, равной 10. Кроме этого, я рассчитываю завершить проект за пять итераций (10 недель). За первую итерацию, однако, мне удалось реализовать не 10, а только пять историй. Если с учетом скорости, которая оказалась в два раза ниже планируемой, скорректировать мое ошибочное представление, то на проект потребуется 10 итераций.

Посмотрим, что произойдет, если провести переоценку. Допустим, я переоцениваю пять реализованных историй и присваиваю каждой из них оценку «два». Моя скорость теперь составляет 10 (пять реализованных историй, каждая по два пункта), а объем оставшейся работы — 45 пунктов. При скорости 10 и оставшихся 45 пунктах я рассчитываю завершить проект за 4,5 итерации. Проблема здесь состоит в смешивании пересмотренной и

первоначальной оценок. Я задним числом переоценил реализованные истории и назначил каждой два пункта. К сожалению, глядя на оставшиеся 45 историй, я не могу предсказать, какие из этих однопунктовых историй мне захочется переоценить задним числом в два пункта.

Когда выполнять переоценку

Продолжим работать с сайтом SwimStats, в этот раз пользовательские истории и оценки приведены в табл. 7.2.

Таблица 7.2. Первоначальные оценки для некоторых историй SwimStats

Номер истории	История	Оценка
1	Как пловец я могу видеть линейную диаграмму своих результатов в конкретных заплывах	3
2	Как тренер я могу видеть линейную диаграмму прогресса в течение сезона для всех моих пловцов в конкретных заплывах	5
3	Как пловец я могу видеть секторную диаграмму, показывающую, сколько раз я занимал первое, второе и третье место	3
4	Как тренер я могу видеть текстовый отчет, показывающий лучшее время каждого пловца в каждом заплыве	3
5	Как тренер я могу загружать результаты соревнований из файла, экспортированного из системы учета времени на соревнованиях	3
6	Как тренер я могу получать от системы рекомендации, кого ставить в какой заплыв, с учетом ограничений на количество заплывов, в которых может участвовать пловец	5

Первые три истории связаны с построением графика для пользователя. Допустим, команда запланировала включить в первую итерацию истории 1, 2 и 6 из табл. 7.2. Ее плановая скорость составляет 13. Однако к концу итерации она реализовала только истории 1 и 6. По словам членов команды, они сделали меньше предполагаемого потому, что история 1 оказалась значительно более трудоемкой, чем ожидалось, и должна оцениваться «как минимум в шесть пунктов». Предположим, что команда недооценила не одну историю, а трудоемкость построения графиков в целом. В этом случае, если история 1 оказалась в два раза более трудоемкой, чем ожидалось, мы должны быть готовы повысить также и трудоемкость историй 2 и 3.

Рассмотрим три сценария развития событий.

Сценарий 1: переоценка не производится

При таком сценарии мы оставляем оценки неизменными. Команда достигает скорости восемь пунктов в завершенной итерации. Это позволяет нам ожидать, что она будет иметь скорость в среднем на уровне восьми пунктов и в последующих итерациях. Вместе с тем команда знает, что не может реализовать истории 2 и 3 за одну итерацию, хотя они и оцениваются всего в восемь пунктов. Поскольку каждая из этих историй связана с построением графиков и ожидается, что каждая связанная с графиками история должна быть в два раза более трудоемкой по сравнению с текущей оценкой (как и история 1), команда приходит к выводу о невозможности реализации историй 2 и 3 за одну итерацию. Они оцениваются в восемь пунктов, но это слишком много.

Сценарий 2: производится переоценка завершенной истории

Посмотрим, поможет ли переоценка одной лишь истории 1 устранить проблему. После завершения итерации команда понимает, что история 1 оказалась в два раза более трудоемкой, чем ожидалось первоначально. Поэтому она принимает решение повысить оценку этой истории до шести. Это означает, что скорость в предыдущей итерации составила 11 — шесть пунктов для истории 1 и пять пунктов для истории 6.

Поскольку другие истории не переоцениваются, команда планирует реализовать в следующей итерации истории 2, 3 и 4. Эти истории оцениваются в 11 пунктов, т.е. представляют такой же объем работы, который был выполнен в предыдущей итерации. Однако команда сталкивается с той же проблемой, что и в первом сценарии: истории 2 и 3 потребуют, скорее всего, в два раза больше времени, чем ожидалось, и выдержать среднюю скорость на уровне 11 пунктов на итерацию не удастся.

Сценарий 3: осуществление переоценки при изменении относительного размера

В этом сценарии команда переоценивает каждую историю, связанную с построением графиков. Оценки историй 1, 2 и 3 удваиваются по сравнению с тем, что показано в табл. 7.2. Как и во втором сценарии, скорость первой итерации составляет 11 — шесть пунктов для истории 1 и пять пунктов для истории 6. Поскольку в первой итерации скорость была равна 11, команда ожидает, что будет держаться на этом уровне и в следующей итерации. Вместе с тем при планировании следующей итерации она выбирает только

историю 2. Эта история, первоначально оцененная в пять пунктов, получает оценку 10 пунктов и оказывается настолько большой, что не оставляет места для дополнительной истории.

Переоценка приносит пользу только в третьем сценарии. Это означает, что переоценивать историю нужно только тогда, когда меняется ее относительный размер.

Переоценка частично реализованных историй

Потребность в переоценке может возникнуть, когда команда реализовала только часть истории в процессе итерации. Допустим, команда работает над историей, которая выглядит следующим образом: «Как тренер я могу получать от системы рекомендации, кого ставить в какой заплыв». Эта история первоначально оценивалась в пять пунктов, но оказалась неожиданно сложной.

Команды на соревнованиях по плаванию получают очки на основе занятых пловцами мест. Однако планирование участия в соревнованиях требует не просто выставления лучшего пловца во всех заплывах. Существует ограничение на количество заплывов, в которых пловец может участвовать. Это означает, что мы не можем выставить Саванну в заплыве на 100 м на спине, поскольку она более полезна в стометровке брассом. Будем считать, что команда достигает конца итерации и система может оптимизировать распределение пловцов по индивидуальным заплывам. Однако команда еще не приступала к обдумыванию процесса выставления пловцов для участия в эстафете. Сколько пунктов необходимо учесть при определении скорости в текущей итерации? Сколько пунктов необходимо присвоить оставшейся работе?

Сначала подчеркну, что при определении скорости я обычно придерживаюсь подхода «всё или ничего»: если история реализована (запрограммирована, протестирована и принята владельцем продукта), то команда получает все пункты, но если в истории что-то осталось недоделанным, она не получает ничего. Если команда предполагает реализовать оставшуюся часть истории в следующей итерации, то такой подход работает хорошо. Ее скорость в первой итерации будет немного ниже ожидаемой, поскольку она ничего не зарабатывает на частично реализованной истории. Во второй итерации, однако, скорость команды будет выше ожидаемой в силу того, что она получит все пункты, хотя часть работы выполнена до начала этой итерации. Это работает хорошо до тех пор, пока все помнят, что нам важно поддерживать стабильную скорость команды на протяжении всего времени, а не учитывать скачки или падения

скорости в отдельно взятой итерации.

Вместе с тем в некоторых случаях нереализованную часть истории невозможно завершить в следующей итерации. В такой ситуации лучше позволить команде получить пункты за завершённую часть истории. Оставшаяся часть (которая является разделом первоначальной истории) переоценивается на основе текущих знаний команды. В нашем случае первоначальная история была оценена в пять пунктов. Если команда считает, что завершённая часть (распределение пловцов по индивидуальным заплывам) эквивалентна трем пунктам или идеальным дням, то она получает именно столько. Незавершённую часть первоначальной истории в нашем случае можно переформулировать так: «Как тренер я могу получать от системы рекомендации, кого ставить в какую эстафету». После этого команда может оценить эту небольшую историю относительно других историй. Суммарная оценка не обязательно должна быть равна первоначальной оценке в пять пунктов.

Так или иначе, лучше не назначать пункты не полностью реализованным историям, а обходиться без таких историй и использовать истории, которые достаточно малы, чтобы проблема частичного получения пунктов за выполненную работу не возникала.

Цель переоценки

Не слишком увлекайтесь процессом переоценки. Когда выясняется, что одна или несколько историй оценены неправильно относительно других историй, старайтесь переоценивать как можно меньше объектов, ограничиваясь лишь приведением относительных оценок в соответствие. Используйте переоценку для учебы и накопления полезного опыта, который пригодится вам при оценке будущих пользовательских историй. Как говорил мне Том Поппендик [\[5\]](#), «неспособность учиться — единственная настоящая ошибка». Учитесь на каждой переоценке истории и приближайтесь к успеху.

Резюме

Четкое понимание того, что пункты и идеальные дни характеризуют размер функции, помогает определить, когда необходима переоценка. Переоценку следует проводить только тогда, когда, по вашему мнению, изменяется относительный размер одной или нескольких историй. Не проводите переоценку только потому, что прогресс идет не так быстро, как вы

ожидали. Пусть скорость, великий уравниватель, позаботится об устранении большинства неточностей оценки.

В конце итерации я не рекомендую давать часть заработанных пунктов за частично реализованные пользовательские истории. Я предпочитаю, чтобы команда учитывала полную оценку при определении скорости (если функция полностью реализована и принята владельцем продукта) или не получала ничего (в противном случае). Вместе с тем команда может принять решение переоценивать частично реализованные пользовательские истории. Обычно это означает оценку пользовательской истории, представляющей работу, которая была выполнена в течение итерации, и одной или нескольких пользовательских историй, которые описывают оставшуюся работу. Сумма этих оценок не обязательно должна быть равна первоначальной оценке.

Вопросы для обсуждения

1. Как скорость корректирует неправильные оценки?
2. Почему переоценку следует проводить только тогда, когда изменяется относительный размер? Приведите несколько примеров из текущего или прошлого проекта, в котором изменялся относительный размер одной или нескольких функций.

Глава 8

Что выбрать — пункты или идеальные дни

Если вы скажете людям, куда нужно добраться, но не укажете пути, то результаты будут поразительными.

Генерал Джордж Паттон

Как показатели размера пункты и идеальные дни имеют свои достоинства. Чтобы помочь вам в выборе того или другого, в этой главе приводятся доводы в пользу каждого подхода.

Доводы в пользу пунктов

Ниже приводится перечень основных доводов в пользу выбора пунктов для оценки размера:

- Пункты способствуют выработке кроссфункционального поведения.
- Оценки в пунктах не устаревают.
- Пункты — это чистый показатель размера.
- Оценка в пунктах обычно требует меньше времени.
- Мои идеальные дни — это не ваши идеальные дни.

Пункты способствуют выработке кроссфункционального поведения

Одна из причин успеха agile-команд заключается в том, что они кроссфункциональны. Иначе говоря, agile-команды состоят из представителей всех дисциплин, необходимых для создания продукта, включая программистов, тестировщиков, менеджеров по продукту, дизайнеров пользовательских интерфейсов, аналитиков и администраторов баз данных. Зачастую, когда кроссфункциональная команда только начинает формироваться, некоторые ее члены с трудом отказываются от

своей цеховой принадлежности. Продукт выигрывает от того, что участники проекта смотрят друг на друга сначала как на членов команды, а уже потом как на специалистов, т.е. по принципу «я участник проекта Нара, и я тестировщик», а не «я тестировщик, прикрепленный к проекту Нара». Различие, может быть, и небольшое, но изменение психологической установки существенное.

Оценка в пунктах может помочь команде научиться работать кроссфункционально. Поскольку оценка в пунктах должна быть единой и представлять работу в целом для всей команды, она инициирует активное обсуждение всех затрагиваемых аспектов. Оценка в идеальных днях, в свою очередь, нередко приводит к тому, что специализированные группы прикидывают, сколько времени займет «их часть» истории, а потом суммируют полученные результаты. Например, программисты могут решить, что им нужно три идеальных дня, администратор баз данных — один день, а тестировщик — два дня. После этого истории присваивается оценка «шесть идеальных дней».

Небольшое различие в том, как происходит первое обсуждение истории, постоянно довлеет над процессом ее реализации.

Оценки в пунктах не устаревают

Оценки, выраженные в пунктах, не устаревают значительно дольше, чем оценки в идеальных днях. Оценка в идеальных днях может меняться среди прочего в зависимости от опыта команды, области ее специализации и состава. Для более глубокого понимания причин этого предположим, что программиста, осваивающего новый язык, спрашивают, сколько времени потребуется на создание небольшого приложения. Допустим, он говорит, что пять дней. Теперь поинтересуемся у того же программиста несколько месяцев спустя, сколько ему потребуется на разработку приложения такого же размера и сложности. Он вполне может сказать, что уложится в один день, поскольку приобрел опыт работы с новым языком. Таким образом, у нас возникает проблема, связанная с тем, что два приложения оцениваются по-разному, хотя имеют один и тот же размер.

Но, может быть, измерение скорости в течение продолжительного периода устранил или компенсирует эту проблему? Увы, надежда на это не оправдывается. Все, что мы видим, это стабильная скорость, хотя объем выполненной работы увеличивается. Предположим, что этот программист — единственный член команды и что он работает итерациями продолжительностью одна неделя. В первый раз, когда он разрабатывает это приложение, его оценка составляет пять идеальных дней. Допустим, в его условиях календарный день равен идеальному дню. Он начинает работать над приложением в первый день итерации и заканчивает его на пятый.

Через несколько месяцев, поскольку оценка аналогичного приложения уменьшается до одного идеального дня, программист разрабатывает пять приложений за одну итерацию. Его скорость опять равна пяти, несмотря на то что он делает в пять раз более объемную работу, чем прежде. В некоторых проектах, особенно в тех случаях, когда осваиваются новые технологии или команда не имеет опыта в данной сфере, этот фактор может быть очень значительным.

Обратите внимание на то, что оценки и в пунктах, и в идеальных днях требуют пересмотра при изменении размера в процессе разработки. Вместе с тем, когда команда приобретает опыт работы в той или иной области, в пересмотре нуждаются только оценки в идеальных днях.

Пункты — это чистый показатель размера

Как говорилось во введении к этой части, первым шагом к оценке сроков реализации чего-либо является оценка размера объекта или объема предстоящей работы. Пункты представляют собой *чистый* показатель размера, а идеальные дни — нет. Идеальные дни можно использовать в качестве показателя размера, но с определенными ограничениями. Как уже отмечалось в предыдущем разделе, оценка в идеальных днях меняется по мере изменения квалификации разработчика. Такого не происходит с пунктами, поскольку размер такой, какой есть, и он не меняется. Это очень желательное свойство для любого показателя размера.

То, что пункты являются чистым показателем размера, дает два преимущества. Во-первых, это означает, что у нас есть возможность оценивать истории исключительно по аналогии. Существуют убедительные свидетельства того, что мы намного лучше оцениваем объекты по принципу «это похоже на то», чем определяем абсолютный размер объектов (Lederer and Prasad, 1998; Vicinanza et al., 1991). При использовании идеальных дней мы также можем оценивать по аналогии, однако в этом случае мы мыслим в категориях календарного графика и продолжительности реализации истории.

Во-вторых, поскольку пункты являются чистым показателем размера и абсолютно отвлеченной величиной, при их использовании нет соблазна сравнивать их с чем-то реальным. Команды, которые оперируют идеальными днями, практически неизбежно сравнивают их с фактическими днями. После этого они пытаются найти причины, по которым работа объемом «только» восемь идеальных дней выполняется за 10-дневную итерацию.

Оценка в пунктах обычно требует меньше времени

Команды, которые оценивают истории в пунктах, решают эту задачу быстрее команд, использующих для оценки идеальные дни. Чтобы оценить большое количество историй, необходимо обсуждение дизайна в общих чертах: должны ли мы реализовать это в базе данных? Можем ли мы использовать существующий пользовательский интерфейс? Как это повлияет на средний ярус? Все эти вопросы возникают в тот или иной момент.

Мой опыт показывает, что команды, использующие для оценки идеальные дни, склонны более глубоко прорабатывать данные вопросы, чем команды, проводящие оценку в пунктах. Разница, по всей видимости, связана с тем, что при оценке в идеальных днях высок соблазн заняться анализом индивидуальных задач, необходимых для реализации истории, в то время как необходимо думать о размере истории относительно других историй.

Мои идеальные дни — это не ваши идеальные дни

Представьте, что два бегуна, один быстрый, другой медленный, стоят в начале беговой дорожки. Оба видят ее целиком и оценивают длину в один километр. Они сравнивают эту дорожку с другой, по которой они уже бегали, и приходят к выводу, что она в два раза короче. Обсуждение ими размера беговой дорожки (в данном случае дистанции) абсолютно предметно.

Предположим теперь, что вместо обсуждения длины дорожки эти два бегуна начинают обсуждать время, за которое пробегут ее. Быстрый бегун может сказать: «Это пятиминутная дорожка». Медленный бегун может ответить так: «Нет, чтобы одолеть ее, нужно не менее восьми минут». Конечно, и тот и другой правы, но прийти к согласию они могут только в том случае, если всегда будут обсуждать дорожки с точки зрения времени одного из них (или кого-то другого).

Та же проблема возникает и в случае использования идеальных дней. Вы можете считать, что полностью справитесь с реализацией пользовательской истории за три идеальных дня. Я считаю, что смогу сделать это за пять дней. Возможно, мы оба правы. Но как нам прийти к согласию? Мы можем выбрать в качестве ориентира вашу оценку, поскольку предполагаем, что именно вы будете выполнять эту работу. Однако это может обернуться ошибкой, если к моменту фактического выполнения работы вы окажетесь слишком занятым и задание придется выполнять мне. Я выполню его позже, поскольку оно оценено в три дня, а мне нужно пять дней.

Большинство команд просто игнорируют эту проблему. Это вполне приемлемо, если все разработчики имеют примерно одну и ту же квалификацию или если программисты всегда работают парами, что

помогает сгладить экстремальные расхождения в производительности.

Доводы в пользу идеальных дней

Ниже приводится перечень основных доводов в пользу выбора идеальных дней для оценки размера:

- Идеальные дни легче объяснить за пределами команды.
- Идеальные дни легче использовать для оценки в начальный момент.

Идеальные дни легче объяснить за пределами команды

Идеальные дни понятны на интуитивном уровне — «это количество времени, которое мне потребуется на эту работу, если я буду заниматься только ею». Поскольку это понятно на интуитивном уровне, оценки в идеальных днях легко объяснить другим за пределами проектной команды. Все понимают, что не все до единой минуты рабочего дня посвящаются программированию, тестированию, дизайну или иным образом используются для создания новых функций.

Внешним наблюдателям (а сначала и команде) обычно приходится объяснять идею применения пунктов в качестве показателя размера. Вместе с тем необходимость объяснения смысла пунктов нередко можно использовать как удобный случай представить общий подход к оценке и планированию проекта. Это великолепная возможность познакомить внешних заинтересованных лиц с такими идеями, как конус неопределенности и последовательное повышение точности плана, и показать, как наблюдение за скоростью на протяжении ряда периодов повышает надежность ваших планов.

Идеальные дни легче использовать для оценки в начальный момент

Помимо простоты объяснения другим смысла оценки в идеальных днях самой команде легче начинать оценку с использованием идеальных дней. Если команда выбирает в качестве показателя пункты, то ей довольно трудно дается оценка первых нескольких историй. Без ориентира, такого как рабочий день с девяти до пяти или оцененные ранее истории, команде, использующей пункты, необходимо найти какую-то базу, от которой можно отталкиваться.

К счастью, большинство команд проходят через эту начальную стадию применения пунктов для оценки очень быстро. Обычно в течение часа они

начинают воспринимать пункты так, словно пользовались ими годами. Так или иначе, первые несколько историй доставляют головную боль.

Рекомендации

Сам я предпочитаю пункты. На мой взгляд, преимущества, которые они дают как чистый показатель размера, очень убедительны. То, что пункты способствуют выработке кроссфункционального поведения, очень положительно сказывается на работе команды. Мышление типа «моя часть займет три идеальных дня, на вашу часть потребуется два идеальных дня, так что в целом нам нужно пять идеальных дней» очень сильно отличается от такого мышления, как «в целом эта история имеет примерно такой же размер, как та история, поэтому давайте также присвоим ей пять пунктов». То, что пункты представляют для меня то же самое, что и для вас, в отличие от идеальных дней, является их большим достоинством. Это позволяет двум разработчикам с разной квалификацией или опытом легко договориться о размере объекта, хотя сроки его реализации будут у них разными.

Недостатки пунктов по-настоящему незначительны. Конечно, легче начать оценку с использованием идеальных дней. Однако неудобство работы с туманными пунктами очень быстро исчезает. Суть оценки в идеальных днях определенно легче объяснить посторонним, но вряд ли стоит выбирать что-то из-за легкости объяснения. Более того, легкость понимания сути идеальных дней создает проблемы. В некоторых организациях стараются сблизить продолжительность фактического дня с продолжительностью идеального дня. Концентрироваться нужно не на этом, а на работе. Стремление организации приблизить фактический день к идеальному, помимо прочего, заставляет давать оценку в фактическом времени, хотя его и называют идеальным. Иными словами, идеальный день определяют как «день, в котором я шесть часов уделяю работе, а два часа занимаюсь прочими вопросами».

Иногда я настаиваю, чтобы команда проводила оценку в идеальных днях. Обычно это случается в ситуации, когда команда не может понять, насколько полезно разделение оценки размера и срока. Некоторые настолько привыкают к требованию оценивать все подряд и выдавать точную дату, что переход к оценке в пунктах дается трудно.

Так вот, хотя команде и предлагается возможность проводить оценку в идеальных днях, я постоянно задаю ее членам вопросы вроде такого: «Насколько велик этот объект по сравнению с тем, что мы оценивали пять минут назад?» или «Так эта история чуть меньше той, что мы только что оценивали?». Цель этих вопросов — перевести разговор в более

абстрактную плоскость и переключить внимание на относительный размер историй с обсуждения сроков разработки дизайна экрана, хранимой процедуры и нескольких форм на HTML. В результате команда, начавшая проводить оценку в идеальных днях, может постепенно отделить свои оценки от количества дней.

Резюме

Команда может проводить оценку либо в пунктах, либо в идеальных днях. Каждый из этих вариантов имеет свои преимущества, позволяющие рекомендовать их.

Пункты помогают команде формировать кроссфункциональное поведение. Кроме того, поскольку пункты являются чистым показателем размера, оценка с их применением не требует корректировки по мере того, как команда приобретает опыт или осваивает определенную сферу. Оценка в пунктах нередко требует меньше времени, чем оценка в идеальных днях. Наконец, в отличие от идеальных дней пункты одинаковы для всех членов команды. Когда один член команды оценивает объект в четыре идеальных дня, а другой — в один идеальный день, они оба могут оказаться правыми, но у них нет основы, на которую можно опираться в споре и прийти к единой оценке.

Достоинство идеальных дней заключается в том, что их суть легко объяснить посторонним и с ними легче начать оценку.

Лично я предпочитаю пункты. У оценки историй в пунктах больше преимуществ. Когда команда не может понять, насколько полезно разделение оценки размера и срока, я настаиваю, чтобы она начала с оценки в идеальных днях, а потом склоняю к переходу на пункты. Чтобы добиться этого, я чаще спрашиваю членов команды о том, «насколько велик этот объект по сравнению с тем, что мы уже оценивали», а не о том, «сколько идеальных дней потребуется для реализации этого объекта». Большинство команд даже не замечают перехода, а когда все же обращают на него внимание, оценка в пунктах уже входит в привычку.

Вопросы для обсуждения

1. Какой подход вы предпочитаете — использование пунктов или идеальных дней? Почему?
2. Как внедрить такую практику во всей организации? Какие препятствия вы предвидите и как вы предполагаете справиться с

ними?

Часть III

Планирование на основе стоимости

Перед составлением плана проекта необходимо определить, что именно требуется нашим пользователям. Простого перечисления аспектов, которые, на наш взгляд, желательны для них, с последующим составлением календарного графика разработки необходимых функций недостаточно. Получение наилучшего сочетания функций продукта (объема), календарного графика и затрат требует целенаправленного учета затрат и стоимости пользовательских историй и тем, которые войдут в релиз.

Каждая из четырех глав этой части начинается с описания одного из четырех факторов, которые необходимо учитывать при приоритизации пользовательских историй и тем. Затем мы рассматриваем простые способы моделирования финансовой отдачи истории или темы, включая приемы сравнения полученных величин. После этого вашему вниманию предлагаются два подхода к оценке желательности историй и тем. Часть завершается рекомендациями по разделению крупных пользовательских историй или функций на части, более подходящие для реализации.

Глава 9

Приоритизация тем

Чтобы получить то, чего вы хотите, сначала нужно решить, что именно вам нужно.

Бен Стейн

Редко когда, если такое вообще случается, мы располагаем достаточным временем, чтобы сделать все. Поэтому нам приходится распределять приоритеты. Ответственность за приоритизацию лежит на всей команде, однако процессом руководит владелец продукта. К сожалению, обычно трудно оценить стоимость небольших единиц функциональности, таких как отдельно взятая пользовательская история. Чтобы справиться с этой задачей, индивидуальные пользовательские истории или функции объединяются в *темы*. Истории и темы затем приоритизируются по отношению друг к другу с целью формирования плана релиза. Темы должны подбираться так, чтобы каждая из них определяла отдельный набор ценных для пользователей или клиента функций. Например, при разработке веб-сайта SwimStats у нас могут быть следующие темы:

- Фиксация всех персональных записей и предоставление пловцам возможности их просмотра.
- Предоставление тренерам возможности оптимально распределять пловцов по заплывам и предсказывать результат команды в соревнованиях.
- Предоставление тренерам возможности вводить план тренировок и отслеживать дистанции, пройденные на тренировках.
- Интеграция с распространенными карманными компьютерами, используемыми в бассейне.
- Импорт и экспорт данных.
- Предоставление официальным представителям возможности следить за результатами заплывов и счетом в соревнованиях.

Каждая из этих тем имеет осязаемую ценность для пользователей программного обеспечения, и им можно присвоить определенную денежную стоимость. С помощью исследований мы можем определить, что

поддержка карманных компьютеров предположительно принесет \$150 000 в виде новых продаж. Это можно сравнить с ожидаемыми новыми продажами в объеме \$200 000, если следующая версия позволит оценивать результаты соревнований по плаванию. Затем можно определить приоритетность этих тем. Однако приоритизация представляет собой нечто большее, чем простой учет денежного дохода от каждого нового набора функций.

Факторы приоритизации

Определение стоимости темы — дело сложное, и владельцы продукта в agile-проектах нередко дают туманную и по большей части бесполезную рекомендацию «приоритизировать по коммерческой стоимости». Было бы понятно, если бы советовали использовать номинальную стоимость. Но что такое *коммерческая стоимость*? Чтобы предложить более практичный набор рекомендаций по приоритизации, в этой главе мы рассмотрим четыре фактора, которые необходимо учитывать при расстановке приоритетов разработки новых возможностей:

1. Финансовая стоимость использования функций.
2. Затраты на разработку (и, возможно, поддержку) новых функций.
3. Объем и значимость обучения и нового знания, созданного в результате разработки функций.
4. Величина риска, ликвидированного в результате разработки функций.

Поскольку большинство проектов осуществляются ради экономии или получения денег, первые два фактора зачастую доминируют в дискуссии о приоритетах. Вместе с тем адекватный учет влияния обучения и риска на проект становится критически важным, если нам нужна оптимальная приоритизация.

Стоимость

Первым фактором при приоритизации работы является финансовая стоимость темы. Сколько денег получит или сэкономит организация в результате включения новых функций в тему? Именно это зачастую имеется в виду, когда владельцев продукта просят «определить приоритеты по стоимости для бизнеса».

Нередко идеальным способом определения стоимости темы является оценка ее финансового эффекта в течение определенного периода времени

— обычно следующих нескольких месяцев, кварталов, а может быть, и лет. Это можно сделать, если продукт продается на коммерческой основе, например новый текстовый процессор или калькулятор, представляющий собой часть встроенной системы. Это также можно сделать для приложений, которые будут использоваться в разрабатывающей их организации. В главе 10 «Приоритизация по финансовой отдаче» описываются различные подходы к оценке финансовой стоимости тем.

Оценка финансовой отдачи темы может вызывать затруднения. Для этого обычно определяют ряд новых продаж, среднюю стоимость одной продажи (включая последующие продажи и соглашения на обслуживание), динамику увеличения продаж и т.д. Из-за сложности такого подхода нередко полезно иметь альтернативный метод оценки стоимости. Поскольку стоимость темы связана с ее желательностью для новых и существующих пользователей, для представления стоимости можно использовать нефинансовые показатели желательности. Этому методу посвящена глава 11 «Приоритизация по желательности».

Затраты

Естественно, себестоимость функции представляет собой действенный детерминант ее приоритетности. Многие функции кажутся превосходными до тех пор, пока мы не узнаем их себестоимости. Важным моментом, о котором нередко забывают, является то, что затраты могут изменяться со временем. Добавление поддержки интернационализации сегодня может занять четыре недели, ее добавление через полгода — шесть недель. Так значит, лучше добавить ее сейчас? Возможно. Допустим, мы затратили четыре недели и сделали это сейчас. В течение следующих шести месяцев мы можем затратить еще три недели на изменение первоначальной реализации в результате приобретения дополнительных знаний за эти шесть месяцев. В данном случае нам лучше подождать. Ну а если мы затратили четыре недели сейчас, а позднее обнаружили, что все можно было сделать проще и быстрее? Наилучший способ сократить затраты на изменение — приступить к реализации функции как можно позже, т.е. когда фактически нет времени на изменения.

Темы нередко кажутся ценными, если смотреть на них только с точки зрения времени реализации. Как это ни банально, очень важно помнить, что время — это деньги. Нередко самым действенным способом добиться этого во время приоритизации является примерный перевод пунктов или идеальных дней в деньги. Предположим, вы повышаете заработную плату всех участвовавших в проекте в течение последних 12 недель и получаете суммарные затраты \$150 000. Сюда входят владелец продукта и руководитель проекта, а также все программисты, тестировщики,

администраторы баз данных, аналитики, дизайнеры пользовательских интерфейсов и т.д. В течение этих 12 недель команда реализовала 120 пунктов. Тогда можно сказать, что совокупная себестоимость составляет \$150 000, а себестоимость пункта — \$1250. Допустим, владелец продукта пытается решить, стоит ли включать в следующий релиз функцию размером 30 пунктов. Один из подходов к принятию решения — определить, оправдывает ли новая функция вложения в размере \$37 500 ($30 \times 1250 = 37\,500$).

В главе 10 «Приоритизация по финансовой отдаче» мы более подробно поговорим о себестоимости и приоритизации на основе финансового вознаграждения относительно затрат.

Новые знания

Во многих проектах значительная доля общих затрат времени и средств идет на приобретение новых знаний. Очень важно признавать это и считать фундаментальным аспектом проекта. Новые знания важны потому, что в начале проекта мы никогда не знаем всего, что нам нужно знать к концу проекта. Знания, приобретаемые командой, можно разделить на две группы:

- Знания о продукте.
- Знания о проекте.

Знания о продукте — это знания о том, *что* должно разрабатываться. Это знания о тех функциях, которые должны включаться, и о тех, которые не должны включаться. Чем больше знаний о продукте имеет команда, тем лучше она принимает решения о характере и функциях продукта.

Знания о проекте, в отличие от этого, это знания о том, *как* должен создаваться продукт. Примерами являются знания о технологиях, которые должны использоваться, о квалификации разработчиков, о том, насколько слаженно работает команда и т.д.

Другой стороной приобретения знаний является уменьшение неопределенности. В начале проекта существует неопределенность в отношении того, какие функции должен иметь продукт. Существует также неопределенность, связанная с тем, как мы будем создавать продукт. Лауфер (Laufer, 1996) называет эти типы неопределенности *конечной неопределенностью* и *неопределенностью средств*. Конечная неопределенность снижается путем приобретения дополнительных знаний о продукте, неопределенность средств снижается путем приобретения дополнительных знаний о проекте.

В проекте, где используется каскадный подход, все неопределенности, связанные с тем, что создается, пытаются устранить до решения проблемы

неопределенности того, как это будет создаваться. Именно отсюда проистекает распространенное представление о том, что анализ связан с тем, что создается, а дизайн — с тем, как это создается. На рис. 9.1 представлены каскадный подход и agile-подход к устранению неопределенности.

В левой части рис. 9.1, где показан каскадный подход, направленная вниз стрелка представляет попытку традиционной команды полностью устранить конечную неопределенность в начале проекта. Это означает, что уже до начала разработки неопределенность, связанная с конечной целью, полностью отсутствует. Продукт полностью определен. Стрелка, направленная вправо, показывает, что неопределенность средств (того, как продукт будет создаваться) снижается по мере осуществления проекта. Разумеется, полное предварительное устранение конечной неопределенности недостижимо. Клиенты и пользователи не имеют точного представления о том, что им нужно, до тех пор, пока им не начинают представлять части продукта. После этого они постепенно уточняют свои потребности.

Agile-подход к устранению неопределенности представлен в правой части рис. 9.1. Agile-команды исходят из того, что в начале проекта невозможно полностью устранить неопределенность, связанную с тем, каким должен быть продукт. Необходимо разработать и представить клиентам некоторые части продукта, получить от них обратную связь, уточнить мнения и скорректировать планы. На это требуется время. Пока идет этот процесс, команда узнаёт больше о том, как нужно разрабатывать систему. В результате одновременно снижается и конечная неопределенность, и неопределенность средств, как показано в правой части рис. 9.1.

Я изобразил кривую в правой части рис. 9.1, чтобы показать предпочтительность раннего снижения конечной неопределенности. Почему, спрашивается, я не провел прямую линию или не построил кривую, свидетельствующую о предпочтительности раннего снижения неопределенности средств? Моя линия отражает важность как можно более раннего снижения неопределенности, связанной с тем, каким должен быть продукт. Нет необходимости устранять конечную неопределенность в самом начале (как предполагает традиционный подход), даже при желании это невозможно. Вместе с тем один из самых серьезных рисков в большинстве проектов — это риск создать несоответствующий продукт. Данный риск можно резко снизить через разработку на раннем этапе тех функций, которые быстрее всего позволяют представить или передать работающую программу реальным пользователям.

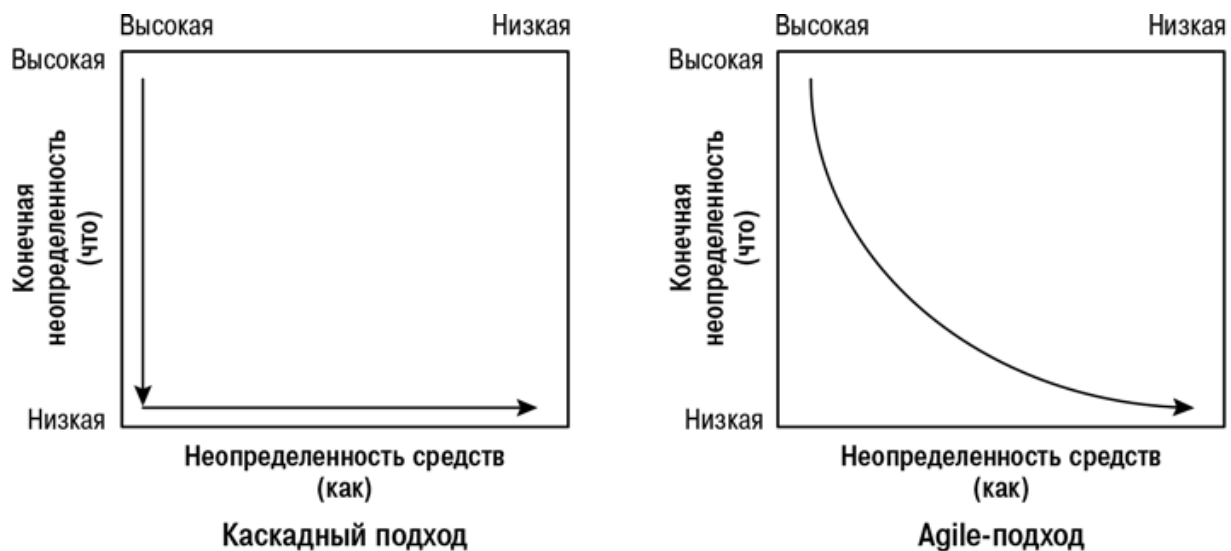


Рис. 9.1. Традиционный подход и agile-подход к снижению неопределенности, основано на работе Лауфера (Laufer, 1996)

Риск

С концепцией нового знания тесно перекликается последний фактор приоритизации: риск. Практически все проекты связаны с очень высоким риском. В наших целях будем понимать под риском все, что пока не произошло, но может произойти и при этом поставить под удар или ограничить успех проекта. С проектами связано множество типов риска, в том числе:

- Риск срыва графика («Мы можем не успеть завершить работу к октябрю»).
- Риск увеличения затрат («Мы можем не найти аппаратные средства с подходящей ценой»).
- Риск функциональности («Мы не обязательно сможем реализовать это»).

Помимо этого, риски можно классифицировать как технологические или деловые.

Классическое противоборство в проекте наблюдается между функциями с высоким риском и функциями с высокой стоимостью. Следует ли проектной команде сначала сконцентрироваться на функциях с высоким риском, которые могут пустить под откос весь проект? Или ей лучше сосредоточить внимание на том, что Том Гилб (Tom Gilb, 1988) назвал «сочными кусками», — на функциях с высокой стоимостью, которые приносят больше всего клиентских долларов?

Чтобы сделать выбор, рассмотрим слабые места каждого подхода. Команда, ориентированная на риск, мирится с тем, что выполняемая ею

работа может оказаться ненужной или имеющей низкую стоимость. Команда может разработать инфраструктурную поддержку для функций, которые окажутся нецелесообразными, поскольку владелец продукта уточняет свое видение на основе того, что он узнаёт от пользователей по мере реализации проекта. В свою очередь, команда, которая фокусируется на стоимости вместо риска, может проделать значительную работу, прежде чем реализовавшийся риск встанет на пути поставки продукта.

Решение, конечно, заключается в том, чтобы не допускать доминирования ни риска, ни стоимости в процессе приоритизации. В целях оптимальной приоритизации важно учитывать и риск, и стоимость. Рассмотрим рис. 9.2, где взаимосвязь между риском и стоимостью функции представлена в четырех квадрантах. Наверху справа находятся функции с высоким риском и высокой стоимостью. Эти функции очень желательны для клиента, однако несут в себе значительный риск разработки. Возможно, функции в этом квадранте основаны на непроверенных технологиях, требуют взаимодействия с непроверенными субподрядчиками, нуждаются в технической инновации (например, в разработке нового алгоритма) или связаны с другими аналогичными рисками. Внизу справа расположены функции, которые в такой же мере желательны, но связаны с меньшим риском. Если функции в правой половине рис. 9.2 очень желательны, то функции, попадающие в левую половину, имеют более низкую стоимость.



Рис. 9.2. Четыре квадранта, представляющие взаимосвязь риска и стоимости

Наиболее целесообразная последовательность разработки функций показана на рис. 9.3. Функции с высокой стоимостью и высоким риском следует разрабатывать в первую очередь. Эти функции приносят наибольшую стоимость, а работа над ними устраняет значительные риски. Следующие на очереди — функции с высокой стоимостью и низким риском. Они приносят такую же стоимость, как и предыдущие, но менее рискованны. Как результат, ими можно заняться позднее. Возьмите за

правило заниматься сначала функциями с высокой стоимостью, а риск используйте в качестве дополнительного фактора.

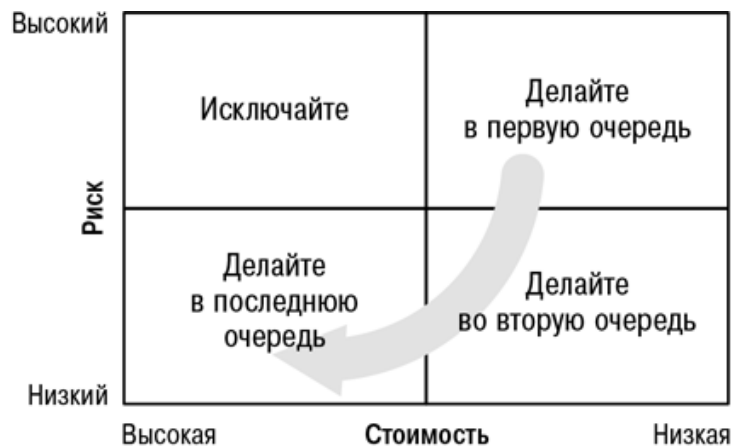


Рис. 9.3. Сочетание риска и стоимости в процессе приоритизации функций

Затем идут функции с низкой стоимостью и низким риском. Ими занимаются в третью очередь, поскольку они меньше влияют на совокупную стоимость продукта в случае отказа от них и связаны с низким риском.

Наконец, отказываться лучше всего от функций с низкой стоимостью, но с высоким риском. Откладывайте работу над всеми функциями с низкой стоимостью, особенно над теми, у которых высокий риск. Старайтесь исключать из проекта объекты с низкой стоимостью и высоким риском. Нет никакого смысла принимать высокий риск в связи с функцией, приносящей незначительную стоимость. Помните, что риск и стоимость функции со временем меняются. Функция с низкой стоимостью и низким риском, стоящая сегодня в квадранте «Исключайте» на рис. 9.3, шесть месяцев назад могла находиться в квадранте «Делайте в первую очередь», если бы все другие функции уже были реализованы.

Объединение четырех факторов

Чтобы объединить все четыре фактора приоритизации, думайте сначала о стоимости функции по сравнению с затратами, которых эта функция потребует при реализации сегодня. Это позволит определить первоначальный порядок реализации тем. Темы с высоким отношением «стоимость/затраты» следует заниматься в первую очередь.

Затем приходит время учета других факторов приоритизации для перемещения тем вперед или назад. Предположим, что на основе стоимости

и затрат тема имеет средний приоритет. Таким образом, команда должна стремиться к реализации этой темы в середине работы над текущим релизом. Вместе с тем технология, необходимая для разработки этой темы, очень рискованная. Этот фактор должен смещать тему вперед по приоритетности и в календарном графике.

Совершенно не обязательно, чтобы такое первоначальное ранжирование с последующим сдвиганием вперед и назад было формализованным видом деятельности. Его может выполнять (и зачастую выполняет) мысленно владелец продукта. Затем он, как правило, представляет свой взгляд на приоритетность команде, которая может попросить владельца продукта немного изменить порядок, исходя из своей оценки тем.

Примеры

Чтобы убедить вас в практичности и полезности этих четырех факторов приоритизации, рассмотрим их применение в двух типичных задачах: создание инфраструктуры и дизайн пользовательского интерфейса. В последующих разделах я представлю тему и покажу, как применять факторы приоритизации.

Создание инфраструктуры

Одна из наиболее распространенных задач приоритизации связана с разработкой инфраструктуры, или элементов архитектуры приложения. В качестве примера возьмем инфраструктуру защиты, которая используется приложением в целом. Если подходить исключительно с точки зрения стоимости, создаваемой для клиентов, то инфраструктура защиты вряд ли попадет в первые итерации проекта. В конце концов, даже несмотря на критическую важность безопасности для многих приложений, в большинстве случаев приложения не приобретают, исходя исключительно из соображений безопасности. Прежде чем возникнет вопрос о безопасности, приложение должно что-то делать.

Следующим фактором приоритизации являются затраты. Добавление инфраструктуры защиты к нашему веб-сайту сегодня будет, возможно, стоить меньше, чем ее добавление на более позднем этапе. Такая ситуация характерна для многих инфраструктурных элементов, и это служит основанием для множества аргументов в пользу их разработки в начале. Вместе с тем, если функция разрабатывается на раннем этапе, есть вероятность того, что она изменится к концу проекта. Затраты на эти изменения необходимо учитывать в любом случае, независимо от того,

когда будет разрабатываться функция, сейчас или позднее. Кроме того, внедрение инфраструктуры защиты на раннем этапе может увеличить сложность, которая представляет собой скрытые затраты для всей будущей работы. Это также необходимо учитывать.

Следующий фактор говорит, что мы должны ускорять разработку функций, которые генерируют новые знания о продукте или проекте. Зависящая от создаваемого продукта инфраструктура защиты вряд ли будет генерировать новые знания о продукте. Так, несколько лет назад мне пришлось работать над проектом, где нужно было аутентифицировать пользователей через LDAP-сервер. Никто из разработчиков не сталкивался с такой задачей ранее, поэтому объем необходимых усилий характеризовался высоким уровнем неопределенности. Для ее устранения истории об LDAP-аутентификации пришлось переместить вверх примерно до средней части проекта, а не оставлять их на самый конец.

Последним фактором приоритизации является риск. Существует ли такой риск, связанный с успехом проекта, который можно устранить путем более раннего внедрения функции защиты? В данном примере, возможно, нет. Так или иначе, отсутствие инфраструктуры, ключевого компонента или другого элемента нередко является существенным риском для проекта. Этого может оказаться вполне достаточно для перемещения разработки ближе к началу по сравнению с тем этапом, на который указывает приоритизация исключительно по стоимости.

Дизайн пользовательского интерфейса

Agile-рекомендация общего характера заключается в том, что дизайн пользовательского интерфейса должен выполняться в пределах той итерации, в которой разрабатывается базовая функция. Это, однако, иногда противоречит аргументам в отношении того, что удобство и простота использования системы повышаются, когда дизайнерам дают возможность заранее обдумать общий вид пользовательского интерфейса. Что мы можем узнать в результате применения наших факторов приоритизации?

Во-первых, будет ли разработка пользовательского интерфейса генерировать значительные, полезные новые знания? Если да, то мы должны переместить часть работы вперед в календарном графике. Да, во многих случаях разработка некоторых компонентов пользовательского интерфейса или навигационной модели приносит значительные, полезные новые знания о продукте. Ранняя разработка некоторых элементов пользовательского интерфейса позволяет показать систему в предварительной форме реальным или потенциальным пользователям. Обратная связь от этих пользователей даст новые знания о продукте, и, опираясь на такие знания, команда может убедиться в том, что она

разрабатывает наиболее ценный продукт.

Во-вторых, приводит ли разработка пользовательского интерфейса к снижению риска? Скорее всего, она не устранил технический риск (если только это не первый опыт работы команды с данным типом пользовательских интерфейсов). Вместе с тем ранняя разработка функций, которые позволяют продемонстрировать пользовательский интерфейс, зачастую снижает самый серьезный риск большинства проектов: риск разработки несоответствующего продукта. Высокий приоритет функций, позволяющих продемонстрировать значительную видимую пользователям функциональность, дает возможность получить отклики пользователей на более раннем этапе. Это наилучший способ избежать риска создания несоответствующего продукта.

Наконец, если затраты на разработку пользовательского интерфейса значительно ниже на раннем этапе, это должно быть еще одним аргументом в пользу перемещения таких функций вперед в календарном графике. В большинстве случаев, правда, о пользовательском интерфейсе такого сказать нельзя.

Таким образом, дополнительные знания и снижение риска дают основания для перемещения на более ранние этапы тех тем, которые позволяют пользователям предоставить обратную связь в отношении удобства и простоты использования, а также функциональности системы. Это не означает, что работа над пользовательским интерфейсом будет осуществляться в изоляции или отдельно от функциональности, лежащей в его основе. Скорее, это говорит о целесообразности перемещения вперед в календарном графике функций со значительными компонентами пользовательского интерфейса, которые позволяют нам получить наиболее полезные отклики клиентов и пользователей.

Резюме

Поскольку мы редко располагаем временем достаточным, чтобы выполнить все, нам необходимо определить, что следует сделать в первую очередь. В процессе приоритизации необходимо учитывать четыре следующих основных фактора:

1. Финансовая стоимость использования функций.
2. Затраты на разработку (и, возможно, поддержку) новых функций.
3. Объем и значимость обучения и нового знания, созданного в результате разработки функций.
4. Величина риска, ликвидированного в результате разработки функций.

Чтобы объединить все четыре фактора приоритизации, думайте сначала о стоимости и затратах на тему. Это позволяет определить первоначальный порядок реализации тем. Темы затем перемещаются вперед или назад с учетом других факторов приоритизации.

Вопросы для обсуждения

1. Функция в проекте, разработку которой вам поручили, имеет довольно низкий приоритет. В настоящий момент кажется, что ее нужно включить в текущий релиз, но в случае нехватки времени от нее можно отказаться. Функцию необходимо разработать на языке, с которым никто в вашей команде не знаком. Что вы сделаете?
2. Какие типы знаний о продукте и проекте ваша команда приобрела с начала нынешнего проекта? Существуют ли дополнительные знания, которые необходимо получить быстро и которые необходимо учитывать при определении приоритетов?

Глава 10

Приоритизация по финансовой отдаче

Когда выгоды невозможно оценить количественно, в качестве общего правила считайте, что их нет.

Том Демарко и Тимоти Листер

Одни проекты реализуются, чтобы получить доход, другие — чтобы снизить расходы, а третьи — чтобы добиться и того и другого. Если можно оценить количество денег, которые принесет или сэкономит каждая тема, это можно использовать в целях приоритизации. Ответственность за прогнозирование финансовой стоимости темы лежит на владельце продукта, однако ее можно разделить с членами команды — программистами, тестировщиками, аналитиками, руководителями проекта и т.д. В большинстве случаев владелец продукта также должен опираться на знание бизнеса и прогнозы групп продаж и маркетинга.

Для определения финансовой отдачи темы я предпочитаю организовывать совещания с участием такого количества людей, которое кажется целесообразным. Целью подобного совещания по оценке темы является заполнение формы, приведенной в табл. 10.1, по каждой теме. В зависимости от количества тем и участников это мероприятие может не ограничиваться одним совещанием.

В табл. 10.1 предусмотрена строка для каждого квартала следующих двух лет. Временной горизонт устанавливается по усмотрению команды. Иногда команды предпочитают определять месячную отдачу для одного или двух лет. По моему опыту, для большинства проектов подходит двухлетний период. Это своего рода золотая середина между гаданием относительно отдаленного будущего и разумным взглядом вперед. Из-за высокой неопределенности, связанной с проектами по разработке программного обеспечения, такого подхода придерживаются и другие (Bills, 2004a).

Таблица 10.1. Рабочий лист для определения отдачи тем

Квартал	Новый доход	Прирост дохода	Сохраненный доход	Операционная эффективность
1				
2				
3				
4				
5				
6				
7				
8				

Табл. 10.1 содержит колонки для различных типов отдачи, которые могут быть у тем. Если в вашем проекте фигурируют другие типы отдачи, измените заголовки соответствующим образом. Аналогичным образом используйте другие заголовки колонок, если требуется их конкретизация (например, «Увеличение дохода от клиентов из США» и «Увеличение дохода от клиентов из Европы»). Совершенно не обязательно иметь для всех тем одинаковый набор колонок.

Участники совещания заполняют рабочий лист, оценивая стоимость в ячейках, которые, по их мнению, затрагиваются в результате разработки темы. Пример заполненного рабочего листа отдачи тем приведен в табл. 10.2. В этом случае включение темы в продукт привлекает новых клиентов («Новый доход»), а также приводит к увеличению дохода от существующих клиентов («Прирост дохода»). Новая тема не оказывает влияния ни на сохранение дохода, ни на операционную эффективность.

Таблица 10.2. Пример рабочего листа отдачи тем

Квартал	Новый доход, \$	Прирост дохода, \$	Сохраненный доход, \$	Операционная эффективность, \$
1	25 000	20 000	0	0
2	35 000	30 000	0	0
3	50 000	40 000	0	0
4	70 000	60 000	0	0
Итого	180 000	150 000	0	0

Откуда берутся цифры? В идеале — из маркетингового исследования, которое использовалось в экономическом обосновании целесообразности проекта. Как минимум тот, кто запрашивает эту тему, должен

количественно оценить основания для ее разработки.

Нельзя сравнивать проекты и принимать решения о приоритетах, просто суммируя числа в строке «Итого» рабочего листа вроде того, что показан в табл. 10.2, для каждой темы. Поток доходов, который составляет \$100 000 в первом квартале, \$200 000 во втором и \$500 000 в третьем, имеет значительно меньшую ценность, чем такой же поток, в котором доходы изменяются в обратном порядке. Для сравнения множества тем нам необходим один или несколько стандартных финансовых показателей. В этой главе мы рассмотрим следующие показатели:

- Чистая приведенная стоимость.
- Внутренняя ставка доходности.
- Срок окупаемости.
- Дисконтированный срок окупаемости.

Однако, прежде чем рассматривать эти финансовые показатели, нам необходимо понять, как проекты приносят или экономят деньги.

Источники дохода

Доход от проекта может поступать из различных источников. Для удобства мы обозначим их как новый доход, прирост дохода, сохраненный доход и операционную эффективность. Хотя в некоторых проектах доминирует какой-либо один источник, большинство проектов имеют несколько источников дохода.

Новый доход

Можно не сомневаться в том, что самым распространенным источником дохода в проекте является получение *нового дохода*. Мало какие компании довольствуются существующей рыночной долей, в большинстве они стараются привлекать новых клиентов. Даже если программный продукт не продается напрямую, добавление новых функций может привести к получению нового дохода. Я, например, работал в компании, которая разрабатывала программное обеспечение для собственного использования при обслуживании своих клиентов-больниц. В какой-то момент наш генеральный директор понял, что при небольшой доработке это программное обеспечение может применяться для предоставления тех же услуг компаниям по медицинскому страхованию. Мы внесли изменения и получили в результате этого совершенно новый источник дохода.

Прирост дохода

Зачастую полезно отличать доход от новых клиентов от дополнительного, приростного дохода от существующих клиентов. *Прирост дохода* может быть результатом того, что новая система или продукт:

- Создает стимулы для приобретения дополнительных лицензий существующими клиентами.
- Содержит опциональные модули расширения, которые можно продавать отдельно.
- Содержит функции, которые позволяют устанавливать более высокую цену.
- Создает стимулы для использования консультационных услуг (например, по интегрированию с приложением стороннего производителя).

Сохраненный доход

Особняком от нового дохода и прироста дохода стоит сохраненный доход. Под *сохраненным доходом* понимается доход, который организация может потерять, если не реализует проект или тему. Допустим, вы успешно продаете продукт для записи пациентов на прием в медучреждение, где работает только один мануальный терапевт. У некоторых ваших клиентов дела идут хорошо, и они увеличивают штат мануальных терапевтов до двух или трех. Если ваше программное обеспечение не будет модифицировано и не сможет поддерживать запись к нескольким мануальным терапевтам, то с растущей практикой вы начнете терять клиентов. Проект по добавлению такой возможности позволит компании сохранить свой доход. Интересно заметить, что в этом случае может также возникнуть потенциал прироста дохода, поскольку у вас появляется шанс повысить цену версии, поддерживающей несколько мануальных терапевтов.

Операционная эффективность

Ни одна организация не может похвастаться предельно возможной эффективностью. Всегда есть задача, которую можно упростить или ликвидировать. Если вы разрабатываете программное обеспечение для использования внутренними клиентами, то вам, скорее всего, хорошо известно о важности операционной эффективности. Вместе с тем, даже когда вы работаете над коммерческой программой, которая будет продаваться за пределами вашей компании, некоторые задачи в рамках проекта могут повышать операционную эффективность. В нашем случае, однако, чаще всего это относится к вашей собственной эффективности.

Например, в ряде проектов, в которых я участвовал, было принято решение разработать собственный инструмент объектно-реляционного отображения для упрощения отображения объектов в таблицах реляционных баз данных. Практически во всех проектах, к которым я имел отношение, создавалась та или иная форма инструмента, облегчающего разработку программного обеспечения.

Зачастую побудительным мотивом повышения операционной эффективности является ожидаемый рост. Неэффективность, которая может не слишком беспокоить вас сегодня, быстро превращается в проблему, когда компания становится значительно крупнее. Предположим, например, что ваша компания создала веб-сайт по продаже рам для картин. Вы предлагаете на продажу не только огромный набор стандартных рам, но и рамы, изготовленные на заказ. Бизнес развивается хорошо, и компания ожидает значительного роста в течение ближайших двух лет. Точнее, она ожидает десятикратного роста продаж за этот период. Как и в любом бизнесе, определенный процент проданных товаров возвращается компании. Автоматизация учета возвращенного товара никогда не имела высокого приоритета, и в настоящий момент возвратом занимается один человек в течение двух часов в день, включая учет запасов и перечисление средств на кредитную карту покупателя. Два часа, потраченные на этот процесс, могут не представлять проблемы сегодня. Однако с увеличением объема продаж на 1000% объем возврата, скорее всего, также вырастет на 1000%, и на его обработку потребуется уже 20 человеко-часов в день. В такой ситуации есть все основания задуматься, не приводит ли этот процесс к неэффективности, которую необходимо устранить.

При поиске возможностей повышения операционной эффективности следует обращать внимание на следующее:

- Все, что занимает много времени или будет занимать много времени, если компания вырастет.
- Улучшение интеграции или коммуникации между отделами.
- Снижение текучести кадров.
- Сокращение времени обучения новых сотрудников.
- Любые чувствительные к срокам выполнения процессы.
- Объединение нескольких процессов.
- Все, что повышает точность и уменьшает переделку.

Пример: WebPayroll

Используя описанные выше приемы, оценим доходы от проекта. Предположим, что наша компания WebPayroll предлагает онлайн-услугу

систему расчета заработной платы компаниям, размеры которых не позволяют им самостоятельно заниматься расчетом налога на фонд заработной платы, распечаткой чеков и т.п. Наши дела идут довольно хорошо, но мы совершенствуем программное обеспечение для сокращения длительности цикла обработки.

В текущий момент мы говорим клиентам, что они должны вводить информацию о заработной плате на наш веб-сайт за три дня до момента выдачи чеков работникам. Наша цель в новой системе — предложение услуги с исполнением на следующий день. Если информация о заработной плате будет введена на сайт WebPayroll до 17:00, то мы сможем сгенерировать чеки, распечатать их и поставить заказчику на следующий рабочий день. Чеки будут вручены клиентам на следующее утро.

Прежде чем начинать оценку дохода от проекта обслуживания на следующий день, необходимо определить, когда он будет реализован. Допустим, разработчики уже оценили истории в этой теме в 150 пунктов. При исторической скорости команды, равной 20 пунктам за двухнедельную итерацию, разработка темы займет $150 / 20 = 7,5 \approx 8$ итераций. Это означает, что повышения дохода и операционной эффективности можно ожидать только после восьмой итерации (если, конечно, нам не удастся поставить частичное решение, что всегда должно быть одной из целей).

Определение нового дохода

Предложение обслуживания на следующий день вместо обслуживания через три дня открывает возможности получения нового дохода. Чтобы оценить эти возможности количественно, оценим сначала число новых клиентов, которых мы сможем приобрести. Надежных данных у нас нет. Однако наш торговый представитель Терри говорит, что примерно треть клиентов, с которыми она общается, отказываются от WebPayroll из-за необходимости подавать информацию за три дня. На основе текущих прогнозов по продажам Терри считает, что сможет привлекать в этом году 50 новых клиентов в квартал, а в следующем году — 100 клиентов в квартал. Эти значения включены в колонку «Новые клиенты» табл. 10.3. Хотя функция обслуживания на следующий день появится лишь в середине второго квартала, Терри уверена, что уже в этом квартале привлечет 50 новых клиентов.

Теперь оценим доход на клиента. Это можно сделать, опираясь на данные по существующим клиентам. Нам известно, например, что средний клиент WebPayroll платит нам \$400 в год. Однако мы считаем, что услуга с исполнением на следующий день будет наиболее привлекательна для мелких клиентов, тех, которые платят нам в среднем \$200 в год. На наш взгляд, можно рассчитывать на дополнительные \$100 в год от каждого

такого клиента. Совокупная стоимость каждого нового клиента, таким образом, составляет \$300 в год, или \$75 в квартал. Поскольку обслуживание на следующий день будет доступно на протяжении лишь двух третей второго квартала, доход на клиента в этом квартале пропорционально уменьшается. Эти величины добавляются в колонку «Доход на клиента» в табл. 10.3, которая позволяет определить значения для колонки «Новый доход».

Таблица 10.3. Прогнозный новый доход от проекта WebPayroll

Квартал	Новые клиенты	Доход на клиента, \$	Новый доход, \$
1	0	0	0
2	50	50	2500
3	50	75	3750
4	50	75	3750
5	100	75	7500
6	100	75	7500
7	100	75	7500
8	100	75	7500

Определение прироста дохода

Под *приростом дохода* понимается дополнительный доход, который мы можем получить от существующих клиентов. На основе наших знаний о существующих клиентах — как часто они опаздывают с подачей информации о заработной плате и т.д., мы считаем, что сможем привлекать примерно по 100 клиентов в квартал до тех пор, пока все 400 существующих клиентов не перейдут на обслуживание на следующий день. Что касается новых клиентов, то услуга будет генерировать примерно по \$100 в год, или \$25 в квартал после того, как она появится во втором месяце второго квартала. Для расчета совокупного прироста дохода в квартал на основе этих данных сформирована табл. 10.4.

Таблица 10.4. Прогнозный прирост дохода от проекта WebPayroll

Квартал	Клиенты	Доход на клиента, \$	Прирост дохода, \$
1	0	0	0
2	100	16	1 600
3	200	25	5 000
4	300	25	7 500
5	400	25	10 000
6	400	25	10 000
7	400	25	10 000
8	400	25	10 000

Определение сохраненного дохода

Сохраненный доход — это то, что мы не потеряем из-за неудовлетворенности клиентов нашим продуктом, роста потребностей или иных причин отказа от WebPayroll. У компании в текущий момент нет хорошего показателя, позволяющего отслеживать этот фактор. Нам известно только, что он начинает превращаться в проблему и станет значительно более серьезным в течение следующих нескольких лет.

По нашим оценкам, переход на обслуживание на следующий день позволит предотвращать уход 20 клиентов в квартал в первый год и 40 клиентов в квартал на второй год. Особенно важно, что эти клиенты продолжают пользоваться WebPayroll, невзирая на отсутствие данной функциональности вплоть до второго квартала. Это означает, что выгоды от проекта обслуживания на следующий день начинаются уже в первом квартале, хотя новая услуга появится лишь во втором квартале.

Зная, что каждый существующий клиент приносит \$400 в год, т.е. \$100 в квартал, мы можем рассчитать сохраненный доход, как показано в табл. 10.5.

Таблица 10.5. Прогнозный сохраненный доход от проекта WebPayroll

Квартал	Сохраненные клиенты	Доход на клиента, \$	Сохраненный доход, \$
1	20	100	2000
2	20	100	2000
3	20	100	2000
4	20	100	2000
5	40	100	4000
6	40	100	4000
7	40	100	4000
8	40	100	4000

Определение операционной эффективности

Чтобы проект обслуживания на следующий день был успешным, нам необходимо почти полностью устранить ручное вмешательство в работу системы, существующее сегодня. В текущий момент для работы системы необходим бухгалтер в офисе WebPayroll, который проверяет правильность информации по заработной плате, а затем вручную вводит ее в два этапа. Сегодня этим у нас занимаются два бухгалтера.

Без автоматизации этой операции нам придется включить в штат двух дополнительных бухгалтеров в середине этого года и еще двух в середине следующего года. В результате планового повышения эффективности в рамках проекта по обслуживанию на следующий день мы предполагаем сокращать по одной из этих штатных единиц каждый год.

Бухгалтеры стоят в среднем \$20 000 в год. Каждому из них, кроме того, предоставляется место в офисе, определенное оборудование, программное обеспечение и компенсационные выплаты. В сумме эти дополнительные скрытые расходы составляют примерно еще 50% заработной платы работника. Иными словами, реальная стоимость бухгалтера ближе к \$30 000 в год. Это называют *полной заработной платой*. Количество бухгалтеров, которые не были наняты, и полная заработная плата дают при перемножении операционную эффективность для каждого квартала, как показано в табл. 10.6.

Таблица 10.6. Прогнозная операционная эффективность от проекта WebPayroll

Квартал	Бухгалтеры, в которых отпала нужда	Полная заработная плата, \$	Операционная эффективность, \$
1	0	0	0
2	0	0	0
3	1	7 500	7 500
4	1	7 500	7 500
5	1	7 500	7 500
6	1	7 500	7 500
7	2	7 500	15 000
8	2	7 500	15 000

Определение затрат на разработку

Для завершения формирования инвестиционного профиля проекта WebPayroll по обслуживанию на следующий день нам необходимо оценить ожидаемые затраты на разработку темы. Чтобы сделать это, посмотрим на заработную плату всех участников проекта (табл. 10.7).

Таблица 10.7. Команда проекта WebPayroll

Роль	Годовая заработная плата, ₴	Полная заработная плата, ₴	Полные затраты на итерацию, ₴	Время в проекте, %	Скорректированные затраты на итерацию, ₴
Владелец продукта	50 000	75 000	2 900	100	2 900
Программист	50 000	75 000	2 900	100	2 900
Программист	30 000	45 000	1 700	50	850
Аналитик	40 000	60 000	2 300	100	2 300
Тестировщик	30 000	45 000	1 700	100	1 700
Тестировщик	50 000	75 000	2 900	100	2 900
Итого					13 550

Полная заработная плата в табл. 10.7 рассчитывается путем увеличения заработной платы каждого работника на 50%. Поскольку итерация занимает две недели, полная заработная плата на итерацию составляет $\frac{1}{26}$ величины полной заработной платы. В колонке «Время в проекте» указана доля времени, которая приходится на каждого члена команды в проекте. Все

заняты полностью, за исключением одного программиста. В колонке «Скорректированные затраты на итерацию» приведены затраты на проект по каждому участнику с учетом полной заработной платы и времени участия в проекте. В целом затраты команды на итерацию составляют \$13 550. Округлим эту величину до \$13 500.

Зачастую полезно знать затраты на один пункт (или один идеальный день). Для определения этой величины разделите скорректированные затраты на итерацию на среднюю или ожидаемую скорость команды. Учитывая, что средняя скорость команды WebPayroll составляет 20 пунктов на итерацию, ее стоимость пункта будет равна $13\,500 / 20 = 675$. Эта информация полезна потому, что, если команду спрашивают, во сколько обойдется разработка элемента, оцениваемого в 100 пунктов, она может сразу дать ответ — \$67 500 (100×675).

Затраты команды WebPayroll обобщены в табл. 10.8.

Таблица 10.8. Суммарные затраты команды WebPayroll

Показатель	Затраты, \$
Затраты на пункт	675
Затраты в неделю	6 750
Затраты на итерацию	13 500

Сведение всех статей вместе

Результаты анализа затрат, нового дохода, прироста дохода и операционной эффективности сведены вместе в табл. 10.9.

Таблица 10.9. Прогнозные доходы по проекту WebPayroll

Квартал	Затраты на разработку, ₽	Новый доход, ₽	Прирост дохода, ₽	Сохраненный доход, ₽	Операционная эффективность, ₽	Чистый денежный поток, ₽
1	–87 750	0	0	2 000	0	–85 750
2	–20 250	2 500	1 600	2 000	0	–14 150
3		3 750	5 000	2 000	7 500	18 250
4		3 750	7 500	2 000	7 500	20 750
5		7 500	10 000	4 000	7 500	29 000
6		7 500	10 000	4 000	7 500	29 000
7		7 500	10 000	4 000	15 000	36 500
8		7 500	10 000	4 000	15 000	36 500

Реализацию функции обслуживания на следующий день предполагается осуществить за восемь итераций, или через 16 недель. На первый квартал приходятся затраты 13 недель в размере \$87 750 (13×6750). На второй квартал приходятся затраты еще трех недель в размере \$20 250.

Финансовые показатели

Определив, как можно оценить денежный поток, генерируемый каждой темой, перейдем к методам анализа и оценки этих денежных потоков. В данном разделе мы рассмотрим чистую приведенную стоимость, внутреннюю ставку доходности, срок окупаемости и дисконтированный срок окупаемости. Каждый из этих показателей можно использовать для сравнения дохода от темы. Однако сначала необходимо разобраться с таким важным понятием, как «временная стоимость денег».

Временная стоимость денег

В одном из первых мультфильмов о моряке Попайе один из персонажей, Вимпи, говорил: «Я с радостью заплачу тебе во вторник за гамбургер, съеденный сегодня». Лишь простофиля мог клюнуть на это предложение Вимпи, поскольку деньги сегодня стоят больше, чем в следующий вторник.

Сегодняшнюю стоимость будущих денег мы определяем с точки зрения того, сколько нужно положить в банк сегодня, чтобы получить будущую сумму. Чтобы купить гамбургер стоимостью \$5 в следующий вторник, мне необходимо сегодня положить в банк \$4,99. Сумму, которую я должен инвестировать сегодня, чтобы получить известную сумму в будущем,

называют *приведенной стоимостью*. Скажем, если я могу заработать 10% на вложения и хочу получить \$1,00 через год, то сегодня мне необходимо вложить \$0,91. Другими словами, при ставке 10% \$0,91 — это приведенная стоимость \$1,00 через год. Если бы я мог заработать на мои деньги 20%, то мне сегодня пришлось бы вкладывать только \$0,83.

Процесс приведения будущих сумм к их нынешней стоимости называют *дисконтированием*. Понятно, что процентная ставка, используемая для дисконтирования будущих сумм, критически важна для определения приведенной стоимости. Ставка, по которой организации дисконтируют будущие деньги, называется *альтернативными издержками* и отражает процентный доход, от которого отказываются, осуществляя данное вложение. У всех нас — физических и юридических лиц — есть разные возможности инвестирования денег. Я могу разместить деньги на сберегательном счете в банке или вложить в акции, инвестировать их в недвижимость или положить под матрас. Организации могут инвестировать деньги аналогичным образом или вкладывать их в проекты. Если в прошлых проектах организация обычно зарабатывала 20%, то новые проекты оценивают по такой же ставке — 20%. Альтернативные издержки организации составляют 20%, поскольку вложение средств в новый проект означает, что организация отказывается от возможности инвестировать их в какой-либо другой проект, который принесет 20%.

Чистая приведенная стоимость

Первым показателем, который мы рассмотрим в целях оценки темы, является *чистая приведенная стоимость* (net present value — NPV). Для определения NPV суммируют приведенные стоимости каждой статьи в потоке будущих стоимостей. Для этого используется следующая формула:

$$NPV(i) = \sum_{t=0}^n F_t(1 + i)^{-t}$$

где i — процентная ставка, а F_t — чистый денежный поток периода t .

Чтобы понять, как это работает, вернемся к нашему примеру WebPayroll. Как уже говорилось, ожидаемые затраты на проект обслуживания на следующий день составляют \$108 000, а генерируемые в результате его реализации доходы и экономия обобщены в табл. 10.9 и воспроизводятся в колонке «Чистый денежный поток» табл. 10.10. Колонка «Коэффициент дисконтирования» в этой таблице представляет собой множитель $(1 + i)^{-t}$ из формулы для расчета NPV и отражает величину, на которую будет дисконтироваться будущий денежный поток. Последняя колонка

«Приведенная стоимость» — это произведение чистого денежного потока и коэффициента дисконтирования. Она, например, показывает, что приведенная стоимость \$18 250 в конце третьего квартала года составляет \$16 701. Сумма значений в колонке «Приведенная стоимость» дает совокупную NPV, равную в нашем случае \$46 341.

Таблица 10.10. Определение NPV для WebPayroll

Конец квартала	Чистый денежный поток, \$	Коэффициент дисконтирования (12% годовых)	Приведенная стоимость, \$
1	–85 750	0,971	–83 252
2	–14 150	0,943	–13 338
3	18 250	0,915	16 701
4	20 750	0,888	18 436
5	29 000	0,863	25 016
6	29 000	0,837	24 287
7	36 500	0,813	29 677
8	36 500	0,789	28 813
NPV (12%) = 46 341			

Достоинством чистой приведенной стоимости при ее использовании для сравнения и приоритизации тем является простота расчета и понятность. Главный недостаток NPV заключается в том, что результаты сравнения значений двух разных денежных потоков могут вводить в заблуждение. Допустим, у нас есть два проекта для выбора. Первый проект требует огромных первоначальных вложений, но имеет NPV, равную \$100 000. Второй проект не требует значительных первоначальных вложений и также имеет NPV, равную \$100 000. Понятно, что мы предпочтем вложить средства в тему, которая связывает меньше денежных ресурсов и имеет ту же NPV. На самом деле нужно представить доход от темы в процентах, чтобы можно было сравнивать темы напрямую.

Внутренняя ставка доходности

Внутренняя ставка доходности (internal rate of return — IRR), которую иногда называют *рентабельностью инвестиции* (return on investment — ROI), позволяет представить доход от проекта в процентном выражении. Если NPV показывает, сколько денег можно получить от проекта (в сегодняшней, приведенной стоимости), то IRR говорит о том, как быстро

растут деньги, вложенные в проект. IRR облегчает сравнение проектов, как это видно из табл. 10.11. Какой проект вы бы выбрали?

Таблица 10.11. Сравнение двух проектов
с использованием NPV и IRR

Проект	Инвестиция, \$	NPV, \$	IRR, %
Проект А	200 000	98 682	27
Проект В	100 000	79 154	43

Большинство, надо думать, предпочтут получать 43% на свои вложения, несмотря на то, что у проекта А выше NPV (он к тому же требует более высоких первоначальных вложений). Денежные потоки для этих двух проектов представлены в табл. 10.12.

Таблица 10.12. Денежные потоки
для проектов из табл. 10.11

Год	Проект А	Проект В
0	–200 000	–100 000
1	50 000	50 000
2	75 000	75 000
3	100 000	50 000
4	170 000	50 000

Многие организации определяют *минимально привлекательный уровень доходности* (minimum attractive rate of return — MARR). Финансируются только те проекты или темы, у которых IRR превышает MARR. Для NPV такой порог устанавливать нецелесообразно, поскольку значения NPV сильно зависят от масштаба проекта. В случае введения порогового значения для NPV небольшие (но ценные) проекты никогда не получат одобрения.

IRR определяется как процентная ставка, при которой NPV денежного потока равна нулю. Другими словами, это значение i^* , при котором

$$0 = PV(i^*) = \sum_{t=0}^n F_t(1 + i)^{-t}$$

Формула для расчета IRR сложная, и ее рассмотрение выходит за рамки настоящей книги. К счастью, большинство электронных таблиц содержат удобную для использования функцию определения IRR. Если вы хотите рассчитать IRR вручную, обратитесь к работе Стива Токи (Steve Tockey, 2004), который, пожалуй, лучше других объясняет, как это сделать. Вместе с тем даже при использовании электронных таблиц необходимо помнить о нескольких важных предварительных условиях:

- Первая одна или несколько статей денежного потока должны быть расходами. (Обратите внимание на то, что должна быть как минимум одна такая статья.)
- Как только денежный поток становится положительным, он больше не должен оказаться отрицательным.
- Сумма положительных статей должна быть больше суммы отрицательных — иначе говоря, приход должен покрывать расход.

Поскольку денежный поток темы обслуживания на следующий день на сайте WebPayroll удовлетворяет этим условиям, мы можем рассчитать IRR. Чтобы сделать это в Excel, введите в ячейку такую формулу:

+IRR({0, -85750, -14150, 18250, 20750, 29000, 36500, 36500}).

Числа в фигурных скобках — это денежные потоки для каждого из восьми кварталов. Ноль в начале показывает, что предварительные затраты в первый день проекта отсутствуют (они вполне могут и присутствовать, если WebPayroll купит дополнительные серверы для осуществления проекта). Для проекта обслуживания на следующий день на WebPayroll IRR составляет 12%, т.е. ожидаемый денежный поток эквивалентен получению годового дохода на вложения компании в размере 12%.

Первым преимуществом использования IRR является отсутствие необходимости определить (или, в худшем случае, угадать) ставку дисконтирования для организации, как это требуется при расчете NPV.

Второе преимущество заключается в том, что с помощью IRR можно напрямую сравнивать проекты. Проект с IRR, равной 45%, имеет более высокую рентабельность инвестиций, чем проект с IRR, равной 25%. Впрочем, обычно нельзя использовать для принятия решений только одну IRR. Допустим, проект с отдачей 45% очень маленький, поэтому 45% мы получаем от маленькой инвестиции, тем не менее этот проект связывает критически важного разработчика. Предположим далее, что другой проект с

отдачей 25%, но крупной инвестицией также требует участия того же самого разработчика. В этом случае вы можете заработать больше денег на втором проекте, том, где IRR более низкая.

Еще один пример: вы можете выбрать проект с IRR 45%, который, однако, начинает приносить выдающуюся отдачу только через два года. А проект с IRR 25% начинает приносить деньги уже через год. Если организация не может позволить себе двухлетние инвестиции, то проект с более низкой IRR становится привлекательным.

Первый недостаток IRR связан с тем, что из-за сложности расчета вручную некоторые могут не доверять полученному результату. Второй недостаток IRR заключается в том, что этот показатель можно рассчитать не во всех случаях. Как мы говорили выше, для получения имеющего смысл значения IRR необходимо выполнение трех условий.

Срок окупаемости

С помощью NPV мы можем представить денежный поток в виде одного показателя — приведенной стоимости. Альтернативно денежный поток можно представить в виде процентной ставки — IRR. Третьим вариантом представления денежного потока является время, необходимое для возврата первоначальной инвестиции. Этот показатель называют *сроком окупаемости*. Чтобы понять, как его определяют, в табл. 10.13 приведены расчеты срока окупаемости для проекта обслуживания на следующий день WebPayroll.

Таблица 10.13. Определение срока окупаемости
для проекта обслуживания на следующий день WebPayroll

Квартал	Чистый денежный поток в конце квартала, \$	Нарастающий итог, \$
1	–85 750	–85 750
2	–14 150	–99 900
3	18 250	–81 650
4	20 750	–60 900
5	29 000	–31 900
6	29 000	–2 900
7	36 500	33 600
8	36 500	70 100

В первом квартале WebPayroll вкладывает в проект \$85 750. Во втором квартале компания делает дополнительное вложение в размере \$14 150. В третьем квартале она начинает возвращать вложенные средства и получает

\$18 250. В какой-то момент в седьмом квартале чистые вложения становятся положительными, поэтому говорят, что срок окупаемости проекта составляет семь кварталов.

У срока окупаемости есть два достоинства при использовании в целях сравнения и приоритизации тем. Во-первых, расчет и интерпретация этого показателя просты. Во-вторых, данный показатель характеризует размер и срок существования финансового риска, принимаемого организацией. Чем больше срок окупаемости, тем рискованнее проект, поскольку на протяжении этого срока может измениться что угодно.

Основной недостаток срока окупаемости заключается в том, что он не учитывает временной стоимости денег. Деньги, которые будут получены через три года, оцениваются так же высоко, как и деньги, уплачиваемые сегодня. Дополнительный недостаток срока окупаемости заключается в том, что он не является показателем прибыльности проекта или темы. Срок окупаемости говорит нам о том, что организация вернет свои деньги через семь кварталов, однако не показывает, сколько денег будет получено.

Дисконтированный срок окупаемости

Первый недостаток срока окупаемости устранить довольно легко. Для этого нужно просто умножить каждую статью денежного потока на соответствующий коэффициент дисконтирования. Как это делается, показано в табл. 10.14 для проекта обслуживания на следующий день WebPayroll.

Таблица 10.14. Определение дисконтированного срока окупаемости для проекта обслуживания на следующий день WebPayroll

Конец квартала	Чистый денежный поток, \$	Коэффициент дисконтирования (12% годовых)	Дисконтированный денежный поток, \$	Нарастающий итог, \$
1	-85 750	0,971	-83 252	-83 252
2	-14 150	0,943	-13 338	-96 590
3	18 250	0,915	16 701	-79 889
4	20 750	0,888	18 436	-61 453
5	29 000	0,863	25 016	-36 437
6	29 000	0,837	24 287	-12 150
7	36 500	0,813	29 677	17 527
8	36 500	0,789	28 813	46 340

Как видно из табл. 10.14, нарастающий итог дисконтированного

денежного потока становится положительным в седьмом квартале (точно так же, как и при расчете простого срока окупаемости).

Сравнение отдачи

По мере оценки каждой темы вы накапливаете информацию, которую можно использовать для сравнения тем и принятия решения о распределении приоритетов, которое является целью этого анализа. Результаты оценки нескольких тем можно представить так, как это сделано в табл. 10.15. Такая таблица позволяет организации быстро проанализировать имеющиеся варианты и выбрать для реализации наиболее ценные темы.

В нашем примере тема обслуживания на следующий день имеет наивысшую чистую приведенную стоимость, но при этом наибольший срок окупаемости инвестиций. Тема интеграции партнеров имеет наивысшую рентабельность инвестиций и самый короткий дисконтированный срок окупаемости, но характеризуется самой низкой NPV. Индивидуализированная отчетность имеет самую низкую рентабельность инвестиций. Вместе с тем ее можно объединить с интеграцией партнеров и реализовать с теми же затратами, что тему обслуживания на следующий день. Принятие решения нельзя назвать простым. Владелец продукта и команде необходимо учесть множество конкретных факторов, таких как склонность организации к риску, потребность в коротком сроке окупаемости, доступность ресурсов и другие возможности инвестирования денег.

Таблица 10.15. Различные показатели оценки тем проекта

Тема	Пункты	Затраты, \$	NPV, \$	ROI, %	Дисконтированный срок окупаемости, кварталы
Обслуживание на следующий день	150	101 250	46 341	45	7
Индивидуализированная отчетность	90	60 750	34 533	15	6
Интеграция партнеров	60	40 500	30 013	49	3

Дополнительная информация по экономике проектов

Хотя в этой главе изложены базовые аспекты использования и расчета четырех финансовых показателей, многое осталось нераскрытым. Дополнительную информацию по вопросу экономики софтверных проектов

можно почерпнуть в книге Стива Токи «Рентабельность разработки программного обеспечения: Максимизация доходности инвестиций» (Return on Software: Maximizing the Return on Your Software Investment) (Steve Tocke, 2004). Все четыре финансовых показателя в данной главе и формулы для их расчета позаимствованы из этой книги.

Резюме

Финансовый анализ тем помогает определить приоритеты, поскольку для большинства организаций решающим фактором является количество заработанных или сэкономленных денег. Обычно достаточно прогнозов по доходам и операционной эффективности на следующие два года. Конечно, в случае необходимости вы можете увеличить временной горизонт.

Хорошим подходом к моделированию дохода от темы является анализ дохода, генерируемого новыми клиентами, существующими клиентами в результате покупки дополнительных копий программ или услуг, клиентами, которые могли бы перейти на использование конкурирующего продукта, и получаемого в результате повышения операционной эффективности.

Деньги, заработанные или потраченные сегодня, стоят больше денег, заработанных или потраченных в будущем. Для сравнения с текущей суммой будущие деньги дисконтируют. Под текущей суммой понимают деньги, которые можно разместить в банке или вложить в какой-либо другой относительно надежный инструмент, так чтобы они превратились в будущую сумму в соответствующий момент в будущем.

Четырьмя показателями, пригодными для оценки денежного потока, являются чистая приведенная стоимость, внутренняя ставка доходности (рентабельность инвестиций), срок окупаемости и дисконтированный срок окупаемости. Определяя эти показатели для каждой темы, владелец продукта и команда могут принимать разумные решения об относительной приоритетности этих тем.

Вопросы для обсуждения

1. Если бы вашей организации пришлось выбирать темы, перечисленные в табл. 10.15, каким было бы наиболее приемлемое решение? Почему?
2. Как бы вы смоделировали доход для темы в своем текущем проекте? Подходят ли вам предложенные категории нового дохода, прироста

дохода, сохраненного дохода и операционной эффективности?
Какие категории вы бы использовали вместо названных?

Глава 11

Приоритизация по желательности

Если вам нужно выбрать одну из двух вещей и вы не можете принять решение, то берите обе.

Грегори Корсо

Эти строки я написал в гостиничном номере. Там было довольно чисто, имелся душ, стояли кровать и письменный стол. Каждый из этих атрибутов является базовым, и я ожидаю увидеть их в любом гостиничном номере. Если что-то отсутствует, я просто не останавлиюсь в такой гостинице. В той гостинице кровать была более удобной, чем в большинстве других; кроме того, гостиница отличалась бесплатным доступом к беспроводному интернету, более просторным номером и наличием фитнес-центра. Я по достоинству оценил эти дополнительные особенности, но для меня они не были обязательными условиями, чтобы остановиться там. Вместе с тем чем больше отличительных особенностей вроде этих предлагает гостиница, тем охотнее я в ней останавливаюсь. Наконец, в данной гостинице меня порадовали два аспекта: тренажеры «беговая дорожка» со встроенными телевизорами в фитнес-центре и бесплатная бутылка воды, ежедневно оставляемая в моем номере.

Я могу разбить особенности этого гостиничного номера на следующие категории:

- Обязательные: кровать, душ, письменный стол, чистота.
- Чем больше, тем лучше: удобство кровати, размер номера, разнообразие и количество тренажеров в фитнес-центре.
- Приятные: встроенные телевизоры на тренажерах «беговая дорожка», бесплатная бутылка воды в номере ежедневно.

Модель удовлетворенности клиентов Кано

Разбивка функций продукта на перечисленные выше три категории может в немалой мере помочь в приоритизации работ для нового релиза. Эту

процедуру предложил Норияки Кано, подход которого позволяет нам разделить функции на следующие три категории:

- пороговые, или обязательные, функции;
- линейные функции;
- привлекательные функции.

К пороговым функциям относят те, которые обязательно должны присутствовать в продукте, если мы хотим, чтобы он был успешным. Часто их называют *обязательными* функциями. Применительно к гостиничному номеру это кровать, душ, письменный стол и чистота. Повышение эффективности или количества пороговых функций незначительно влияет на удовлетворенность клиентов. Так, пока душ в гостинице отвечает моим базовым потребностям, мне безразлично, из чего сделана столешница тумбы с раковиной.

Линейными функциями считаются те, для которых справедливо правило «чем больше, тем лучше». Чем просторнее гостиничный номер, тем лучше. Их называют линейными потому, что удовлетворенность клиентов линейно связана с количеством данных функций. Чем лучше эти функции работают (или чем больше их имеется), тем выше будет удовлетворенность клиентов. Из-за этого цена продукта нередко привязывается к линейным атрибутам. Если в гостинице есть гантели и пара хороших тренажеров «беговая дорожка», то я буду более счастлив, чем при наличии лишь одного старинного тренажера «лестница». Я буду еще более счастлив, если обнаружу там еще силовой тренажер и велотренажер. Для гостиницы в этом важно то, что я с большей вероятностью остановлюсь в ней еще раз и заплачу более высокую цену за номер.

Наконец, к привлекательным относят те функции, которые обеспечивают восхищение и нередко готовность заплатить более высокую цену за продукт. Вместе с тем отсутствие привлекательных функций не снижает удовлетворенность клиентов ниже уровня нейтральности. Встроенный в гостиничный тренажер телевизор был для меня привлекательным качеством. Я бы не уехал с чувством неудовлетворенности, если бы его не было, поскольку не встречал такого ни в одной другой гостинице. Фактически привлекательные функции зачастую порождают новые потребности, так как клиенты или пользователи не подозревают о потребности в этих функциях, пока не увидят их.

Взаимосвязь этих трех типов функций представлена на рис. 11.1. Стрелка, идущая снизу вправо на рисунке, показывает, что после реализации определенного количества обязательной функции удовлетворенность клиентов не увеличивается с наращиванием ее объема. Также не имеет значения, в какой мере добавляется обязательная функция, — удовлетворенность клиентов никогда не поднимается выше срединной

ТОЧКИ.

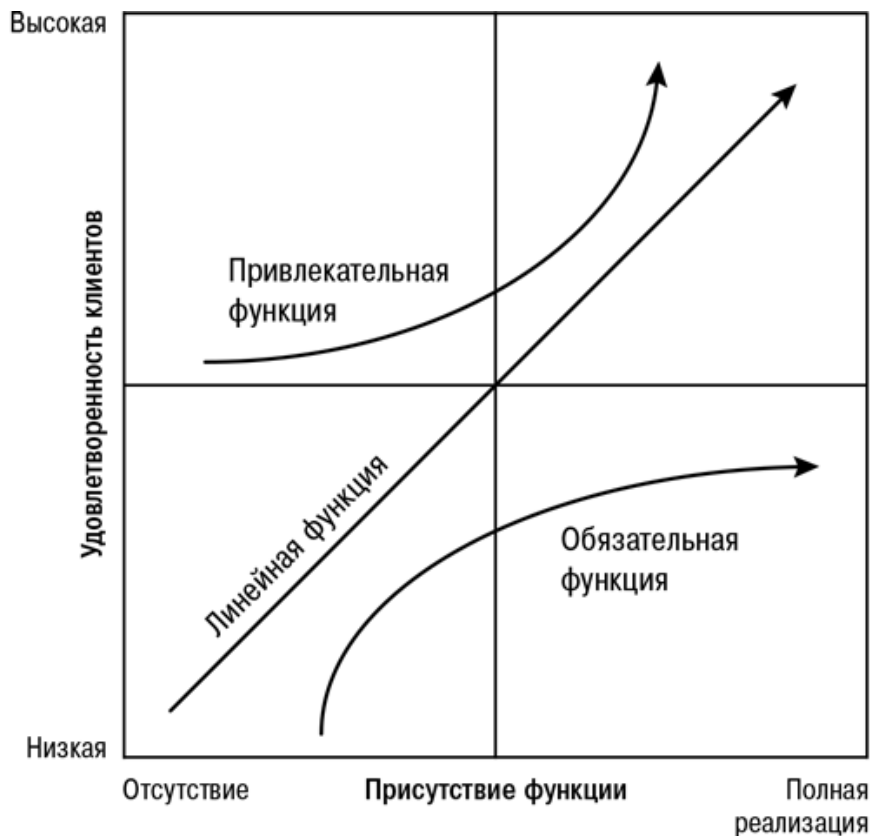


Рис. 11.1. Модель удовлетворенности клиентов Кано

В отличие от этого линия в верхней части слева говорит о том, что удовлетворенность клиентов резко возрастает даже при ограниченной реализации привлекательной функции. Наконец, линия, проходящая посередине, демонстрирует прямую взаимосвязь между включением линейных функций и удовлетворенностью клиентов.

Поскольку обязательные функции необходимы продукту, чтобы занять рыночный сегмент, акцентировать внимание следует на приоритетности разработки всех пороговых функций. Нет нужды разрабатывать обязательные функции продукта в первых итерациях релиза. Вместе с тем в силу того, что пользователи считают эти функции обязательными, они должны быть реализованы до выпуска готового продукта. Не забывайте при этом, что может быть вполне приемлемой частичная реализация обязательных функций, поскольку прирост удовлетворенности клиентов быстро падает после достижения базового уровня поддержки таких функций. Например, я пишу эту книгу в издательской системе Adobe FrameMaker, которая поддерживает минимальную функцию «отмена», обеспечивая возврат лишь на один уровень. С моей точки зрения, ее разработчики продвинулись со своей одноуровневой функцией «отмена» довольно далеко по оси удовлетворенности клиентов, однако я бы предпочел, чтобы этих уровней было намного больше.

Во вторую очередь внимание следует сконцентрировать на реализации как можно большего количества линейных функций. Поскольку каждая из этих функций напрямую повышает удовлетворенность клиентов, чем больше их будет включено, тем лучше (за исключением, конечно, таких ситуаций, когда продукт и без того раздулся от слишком большого количества функций). Наконец, если позволяет время, нескольким привлекательным функциям необходимо присвоить такой приоритет, чтобы они попали в план релиза.

Учтите, что функции на диаграмме Кано со временем смещаются вниз. Еще совсем недавно беспроводной доступ в интернет в гостиничном номере был привлекательной функцией. Сейчас это уже линейная функция, которая вот-вот станет пороговой.

Оценка тем по модели Кано

Самый простой подход к использованию модели Кано в agile-проекте — анализ всех тем и их грамотная разбивка по типам. Намного лучше, однако, определить тип каждой темы совместно с клиентами или пользователями. Это на удивление легко, поскольку можно использовать письменный опросный лист. Для точной приоритизации требований опросный лист, возможно, придется направить 20–30 пользователям (Griffin and Hauser, 1993).

Кано предлагал определять категорию функции с помощью двух вопросов: один вопрос для выяснения того, что пользователь будет чувствовать при наличии функции в продукте, и второй — что он будет чувствовать при ее отсутствии. Говорят, что первый вопрос имеет функциональную форму, поскольку предполагает наличие функции, а второй — дисфункциональную форму, поскольку в этом случае функция отсутствует. Ответ на каждый вопрос выбирается на стандартной шкале из пяти пунктов:

1. Мне это нравится.
2. Я ожидал, что это будет выглядеть так.
3. Мне безразлично.
4. Я могу смириться с этим.
5. Мне это не нравится.

В качестве примера вспомним наш веб-сайт SwimStats. Допустим, мы собираемся добавить три новые функции:

- Возможность видеть график результатов пловца в соревнованиях прошлого сезона.

- Возможность для пловцов размещать информацию о своей специализации.
- Возможность для любого зарегистрированного члена сайта загружать фотографии.

Чтобы определить типы этих функций, мы проводим опрос потенциальных пользователей, задавая им следующие вопросы:

- Если бы была возможность построить график результатов пловца в соревнованиях прошлого сезона, как бы вы к этому отнеслись?
- Если бы не было возможности построить график результатов пловца в соревнованиях прошлого сезона, как бы вы к этому отнеслись?
- Если бы у пловцов была возможность размещать информацию о своей специализации, как бы вы к этому отнеслись?
- Если бы у пловцов не было возможности размещать информацию о своей специализации, как бы вы к этому отнеслись?
- Если бы была возможность загружать фотографии, как бы вы к этому отнеслись?
- Если бы не было возможности загружать фотографии, как бы вы к этому отнеслись?

Первая пара этих вопросов и гипотетические ответы одного из пользователей приведены на рис. 11.2.

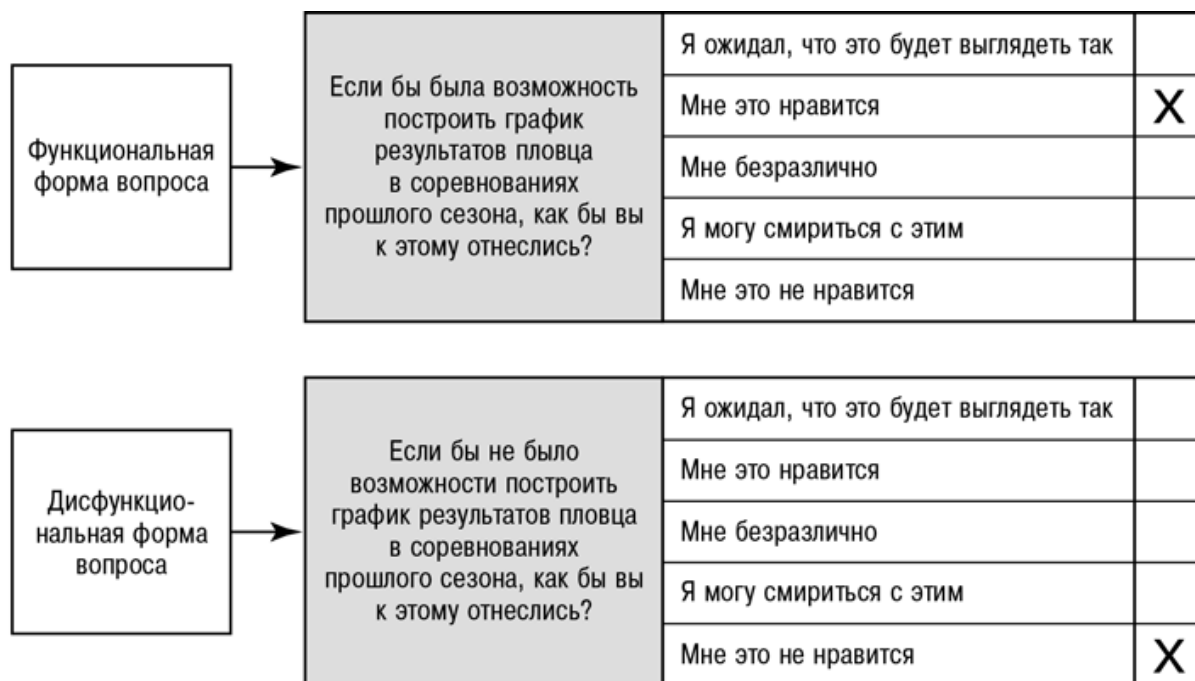


Рис. 11.2. Функциональная и дисфункциональная формы вопросов

Из рис. 11.2 следует, что пользователь вполне может давать взаимоисключающие ответы. Например, он может сказать, что ему

нравится возможность построить график результатов пловца в соревнованиях, и в то же время, что ему не нравится та же самая функция. Существует 25 возможных комбинаций ответов на функциональную и дисфункциональную формы вопроса. Нам необходимо найти способ одновременного просмотра обоих ответов и оценки мнения пользователя.

Категоризация ответов

Такой способ представлен на рис. 11.3. С помощью перекрестной связи ответов на функциональный и дисфункциональный вопросы реакцию потенциального пользователя можно свести к однозначному варианту. Поэтому если пользователь говорит, что он ожидает получить возможность строить график результатов в соревнованиях и что ему не нравится отсутствие такой возможности (как на рис. 11.2), то мы устанавливаем перекрестную связь ответов и получаем, что он считает эту функцию обязательной. По его представлениям, возможность строить график результатов в соревнованиях является обязательной функцией.

		Дисфункциональный вопрос				
		Нравится	Ожидает	Безразличен	Мирится	Не нравится
Функциональный вопрос	Нравится	Q	E	E	E	L
	Ожидает	R	I	I	I	M
	Безразличен	R	I	I	I	M
	Мирится	R	I	I	I	M
	Не нравится	R	R	R	R	Q

М – обязательная R – обратная
 L – линейная Q – сомнительная
 E – привлекательная I – безразличная

Рис. 11.3. Категоризация функции по ответам на пару вопросов

Если повторить этот процесс с 20–30 пользователями, то их ответы можно сгруппировать и определить распределение, как показано в табл. 11.1. Анализ таблицы показывает, что построение графика результатов соревнований является одномерной (линейной) функцией, а загрузка фотографий — обязательной функцией.

Таблица 11.1. Распределение результатов опроса пользователей

Тема	Е	L	M	I	R	Q	Категория
График результатов соревнований	18,4	43,8	22,8	12,8	1,7	0,5	Линейная
Загрузка фотографий	8,3	30,9	54,3	4,2	1,4	0,9	Обязательная
Размещение информации о специализации	39,1	14,8	36,6	8,2	0,2	1,1	Привлекательная Обязательная

Иногда встречаются функции, наподобие размещения информации о специализации в табл. 11.1, которые получают высокие оценки по двум категориям. Это означает, что у разных клиентов и пользователей разные ожидания. В таких случаях анализ ответов необходимо проводить с учетом какого-либо фактора, позволяющего разделить клиентов или пользователей на подгруппы. Можно, например, отделить ответы пользователей из небольших компаний от ответов пользователей из крупных компаний. А можно разделить ответы на основе ролей в компании или в отраслевом сегменте.

Относительное взвешивание: еще один подход

Карл Вигерс (Wiegers, 1999) рекомендует подход, аналогичный подходу Кано, где учитываются и положительное влияние присутствия функции, и отрицательный эффект ее отсутствия. Вместо использования опросного листа этот подход предлагает опираться на экспертную оценку. Команда коллективно под руководством владельца продукта оценивает каждую предлагаемую для включения в следующий релиз функцию. Функция оценивается с точки зрения выгод, которые принесет ее реализация, а также с точки зрения потерь, которые возникнут при ее отсутствии. Как и оценки с использованием пунктов и идеального времени, оценки выгод и потерь являются относительными. Применяется шкала от 1 до 9. В табл. 11.2 на примере функций SwimStats показано, как это осуществляется.

Выгоды от добавления функции построения графика результатов пловца в соревнованиях в течение сезона оценены в 8 баллов. По ценности эта функция лишь немного уступает функции загрузки фотографий, но значительно превосходит функцию размещения информации о специализации. Что касается относительных потерь в случае отсутствия этих функций, то лишение пользователей возможности загружать фотографии оказывается почти незаметным. (В соответствии с подходом Кано загрузка фотографий — скорее всего, привлекательная функция, она

нравится пользователям, но они не откажутся от продукта при ее отсутствии.) Более значительные потери приносит отсутствие возможности построить график результатов в соревнованиях и разместить информацию о специализации.

Таблица 11.2. Приоритизация на основе относительного взвешивания

Функция	Относи- тельные выгоды	Относи- тельные потери	Общая стои- мость	Стои- мость, %	Оценка	Затраты, %	Приори- тет
График результатов соревнований	8	6	14	42	32	53	0,79
Загрузка фотографий	9	2	11	33	21	34	0,97
Размещение информации о специализации	3	5	8	25	8	13	1,92
Итого	20	13	33	100	61	100	

Для каждой функции относительные выгоды и потери суммируются и отражаются в колонке «Общая стоимость». При желании можно определить весá выгод и потерь. Например, чтобы придать относительным выгодам в два раза больший вес, чем относительным потерям, умножьте относительные выгоды на 2 перед их сложением с относительными потерями при вычислении общей стоимости.

Суммарное значение в колонке «Общая стоимость» (в нашем случае 33) представляет стоимость реализации всех функций. Для расчета относительного вклада каждой функции разделите ее общую стоимость на суммарное значение колонки «Общая стоимость». Например, построение графика результатов соревнований имеет стоимость 14. Общая стоимость всех функций составляет 33. Деление 14 на 33 дает 0,42, или 42%. Это означает, что возможность построения графика результатов соревнований представляет 42% общей стоимости всех перечисленных функций.

В следующей колонке, «Оценка», приведена оценка в пунктах или идеальных днях. Как и в случае «Общей стоимости», значения этой колонки суммируются (в нашем случае 61), после чего определяется процентная доля каждой оценки (колонка «Затраты, %»). В этом примере построение графика результатов соревнований оценено в 32 пункта или идеальных дня. Сумма трех оценок составляет 61 пункт или идеальный день. Поэтому возможность построения графика результатов соревнований представляет 53% затрат на все три истории ($32 / 61 = 0,53$).

Значения в последней колонке, «Приоритет», определяются путем

деления стоимости в процентах на затраты в процентах. Более высокие значения представляют более высокие приоритеты, поскольку соответствующие функции создают более значительную стоимость на затраченные на них силы и время. Это ясно видно в табл. 11.2 на примере функции размещения информации о специализации. Данная функция приносит немногим более половины стоимости функции построения графика результатов соревнований (общая стоимость 8 по сравнению с 14), однако на ее реализацию требуется всего одна четверть соответствующих затрат (оценка 8 по сравнению с 32). Из-за высокого отношения стоимости к затратам возможность размещения информации о специализации имеет наивысший приоритет в настоящем анализе.

Важность включения относительных потерь

У вас может возникнуть соблазн удалить колонку «Относительные потери». Не делайте этого. Как в подходе Кано, она может быть очень полезной при рассмотрении функций с точки зрения воздействия на пользователей наличия и отсутствия конкретной функции.

Ситуация у одного из моих клиентов в сфере финансовых услуг очень хорошо демонстрирует важность учета относительных потерь. У владельца продукта не было никаких трудностей с определением приоритетов большинства функций для их команды. Однако одна функция, подлежащая приоритизации, не приносила никакой стоимости компании, кроме обеспечения соблюдения новых правил регулирования деятельности. Владелец продукта шутил, что единственное, к чему может привести отсутствие этой функции, это тюремное заключение для генерального директора. И владельца продукта (поскольку он не был генеральным директором) вполне устраивал такой риск — ведь он позволял включить в продукт другие функции.

Это наглядный пример функции с относительными выгодами, равными 1, и относительными потерями, равными 9.

Резюме

Вернемся на шаг назад и вспомним, зачем мы все это делаем. В предыдущей главе мы говорили о том, что в процессе приоритизации функций необходимо рассматривать следующие четыре фактора:

1. Объем и значимость обучения и нового знания, созданного в результате разработки функций.
2. Величину риска, ликвидированного в результате разработки функций.

3. Финансовую стоимость использования функций.
4. Затраты на разработку (и, возможно, поддержку) новых функций.

В главе «Приоритизация тем» речь шла о важности учета обучения и сокращения риска при приоритизации функции. В настоящей главе представлены два подхода к приоритизации по желательности функции: анализ Кано и относительное взвешивание.

В анализе Кано функции разделяют на обязательные, линейные и привлекательные. Это делается на основе опроса потенциальных пользователей, которым задают два вопроса: как они отнеслись бы к наличию определенной функции и как они отнеслись бы к ее отсутствию.

Относительное взвешивание представляет собой подход к оценке выгод от реализации функции и потерь в результате отказа от ее реализации, а также к приведению затрат к единому значению, представляющему приоритет функции.

Вопросы для обсуждения

1. Назовите сравнительные достоинства анализа Кано и относительного взвешивания в вашей организации.
2. Какие привлекательные функции предполагается включить в ваш текущий проект?
3. Можно ли сказать, что ваш текущий проект имеет оптимальное соотношение привлекательных, линейных и обязательных функций?

Глава 12

Разбивка пользовательских историй

Сегодня мы не пишем программы модуль за модулем; мы создаем программы функция за функцией.

Мэри Поппендик

По мере того как пользовательские истории перемещаются в верхние строки плана релиза, т.е. по мере приближения реализации, нередко возникает потребность в их разбивке. В самом деле, когда реализация конкретной пользовательской истории требует больше времени, чем продолжительность итерации, не остается ничего иного, как разбить историю на две части или более. Научиться разбивать истории — не такая уж трудная задача, однако она требует определенных навыков и опыта. Чем разнообразнее пользовательские истории, которые вы разбивали в прошлом, тем легче вам будет справиться с новыми. С учетом этого в настоящей главе предлагаются рекомендации по разбивке историй на нескольких примерах. Эти примеры помогают сформулировать ряд правил, которые можно применять при разбивке других историй.

Рекомендации из этой главы можно использовать в любой момент при возникновении потребности в разбивке истории. Вместе с тем правила ориентированы прежде всего на пользовательские истории или функции, которые трудно поддаются разбивке.

Когда нужно разбивать пользовательскую историю

Бывают ситуации, в которых пользовательскую историю приходится разбивать на несколько частей. Во-первых, пользовательские истории требуют разбивки, когда они слишком велики и не укладываются в одну итерацию. Иногда история не укладывается в итерацию потому, что просто больше нее. Понятно, что историю в такой ситуации необходимо разбить. Или история меньше планируемой итерации, но не укладывается в нее из-за

недостатка места. Команда может решить, что времени достаточно для разработки за итерацию только части этой истории.

Во-вторых, может быть полезно разбить крупную пользовательскую историю (*эпонею*), если требуется более точная оценка. Приведу пример. Один из моих клиентов рассматривал возможность реализации новых функций, которые должны были обеспечить расширенный доступ в его систему работавшим в других компаниях представителям службы по поддержке клиентов. Первый вопрос, на который нужно было ответить владельцу продукта: стоит ли овчинка выделки? Вместо того чтобы составлять кучу индивидуальных пользовательских историй, он написал одну большую историю и изложил свое видение этой истории команде. Команда оценила ее в 70 пунктов. Это было достаточно хорошо, чтобы владелец продукта захотел добавить новые функции. Он знал, что с оценкой связана большая неопределенность, но, даже если бы ошибка составляла 100%, реализация функций все равно имела бы смысл. Если бы возникла проблема с включением дополнительных 70 пунктов в один релиз, владелец продукта мог бы разбить большую историю на части и предложить команде оценить несколько более мелких историй.

Разбивка по границам данных

Лучше всего разбивать большие пользовательские истории по данным, которые будут поддерживаться. Например, в одном из проектов команда разрабатывала продукт, предназначенный для сбора финансовой информации. Она начала с одной крупной пользовательской истории: «Как пользователь я могу вводить информацию из моего баланса». Баланс в данном случае мог иметь очень много полей. На самом высоком уровне в нем находятся активы и пассивы. Активы содержат такие статьи, как денежные средства, ценные бумаги, объекты недвижимости, автомобили, кредиты. Система позволяла пользователю работать с этим балансом на различных уровнях детализации. Пользователь мог ввести одну сумму, представляющую все его активы. Или он мог ввести более детальную информацию (подробное перечисление всех кредитов). Вывод множества полей на экран и взаимодействие между этими полями представляли задачу значительно более крупную, чем команда, по ее оценкам, могла реализовать за одну двухнедельную итерацию.

Команда разбила эту историю по типам данных, которые мог вводить пользователь. Первая история выглядела так: «Как пользователь я могу вводить суммарные балансовые показатели». Эта история была очень маленькой (даже слишком маленькой), поскольку предполагала создание

лишь базовой формы и двух полей: активы и пассивы. Следующая история была сформулирована следующим образом: «Как пользователь я могу вводить балансовые данные по категориям». Эта история покрывала следующий уровень детализации (денежные средства, ценные бумаги, объекты недвижимости, кредиты и т.д.). В результате реализации этой истории на экране должны были появиться два десятка полей для ввода данных. Очередная история касалась подтверждения правильности данных: «Как пользователь я хочу, чтобы была возможность подтверждения введенных данных во избежание ошибок». Команда обсудила смысл этого и пришла к выводу, что необходимо предоставить возможность ввода положительных и отрицательных чисел, что десятичные знаки могли вводиться, но суммы должны были автоматически округляться до целых чисел и т.д.

Следующая история звучала так: «Как пользователь я могу вводить детальную информацию по кредитам». Эта история должна была позволить пользователю вводить до 100 кредитов (это число обсудили, согласовали и сделали условием удовлетворенности на карточке истории). Данная история была крупнее, чем выглядела, поскольку решала несколько проблем с пользовательским интерфейсом, таких как количество новых рядов данных по кредитам, которые должны отображаться на экране. Эта история оказалась намного крупнее других, которые получились при разбивке исходной истории. Вместе с тем даже она была значительно меньше исходной истории, поскольку касалась поддержки детальных данных только по кредитам. Историю, связанную с кредитами, сделали образцом для многих других пользовательских историй, которые разбивались на части, вроде таких, как «Как пользователь я могу вводить детальную информацию по моим объектам недвижимости» и «Как пользователь я могу вводить детальную информацию по моим денежным авуарам, включая текущие и сберегательные счета».

Разбив исходную историю таким образом, команда создала около десятка пользовательских историй. Каждая новая история имела теперь такой размер, что ее можно было реализовать в течение двухнедельной итерации. Это дает нам первое правило:

Разбивайте большие истории по границам данных, поддерживаемых историей.

Разбивка пользовательской истории по границам данных — очень полезный подход, который обязательно должен быть в вашем портфеле инструментов. Приведу еще один пример. Несколько лет назад я работал с командой, разрабатывавшей автоматизированную подсистему

факсимильной связи. Команда столкнулась с большими пользовательскими историями относительно того, как сконфигурировать систему. Эти истории удалось значительно уменьшить, разделив поддержку внутриамериканских и международных телефонных номеров.

В некоторых случаях большую историю можно значительно уменьшить, удалив из основной истории обработку исключительных или сбойных ситуаций. Допустим, вы работаете над системой для обработки погашений кредитов и имеете такую пользовательскую историю: «Как заемщик я хочу иметь возможность погасить мой кредит». Когда команда обсуждала эту историю, владелец продукта подчеркнул, что в случае нечаянной отправки заемщиком чека, превышающего сумму непогашенного остатка, необходимо распечатать чек на возвращаемую сумму и отправить обратно заемщику. Он добавил, что это относится только к суммам более \$2. Эту историю можно разбить на следующие, более мелкие истории:

- Как заемщик я хочу иметь возможность погасить мой кредит.
Примечание: допускаются переплаты.
- Как заемщик я в случае ненамеренной переплаты получаю возврат излишка, если он превышает \$2.

Разбивка по операционным границам

Не так давно я работал с командой, перед которой стояла задача создать очень сложное поисковое окно. В нем были десятки полей в верхней части, средство формирования запросов среднего яруса, которое собирало сформулированные на основе введенной информации запросы к базе данных, а также сложная таблица для воспроизведения данных в нижней части. Вся эта работа первоначально описывалась одной пользовательской историей. Я порекомендовал команде разбить работу на три части, которые распределялись на три итерации.

В первой итерации команда работала над созданием базового пользовательского интерфейса, включая примерно половину полей с поисковыми критериями, которые располагались в верхней части окна. Она также разрабатывала части средства формирования запросов, которые работали с этими полями. Нижняя часть окна должна была содержать сложную таблицу для воспроизведения данных. Это было слишком много для одной двухнедельной итерации. Поэтому в конце первой итерации в этой части окна отображалось простое сообщение вроде: «В результате этого поискового запроса найдено 312 совпадений». Конечно, подобное сообщение было не слишком полезно для пользователя, который хотел знать, что именно крылось за этими 312 совпадениями. Однако оно

представляло значительный прогресс и делало этот прогресс видимым всем, кто был заинтересован в проекте.

Во второй итерации этого проекта добавлялась таблица для воспроизведения данных, в третьей — все оставшиеся поля для поисковых критериев в верхней части окна. Итерации были приоритизированы таким образом из-за наличия значительной неопределенности в отношении количества времени, необходимого на разработку таблицы для воспроизведения данных. Устранение этой неопределенности во второй итерации сочли более целесообразным, чем сохранение ее до третьей итерации. Поскольку команда уже получила хорошее представление о том, что необходимо для создания средства формирования запросов, по ее мнению, эта работа уменьшала риск. Это подводит нас к следующему правилу:

Разбивайте большие истории на основе операций, которые выполняются в пределах истории.

Общий подход к этому заключается в разбивке истории по границам обычных CRUD-операций — создание, чтение, редактирование и удаление. Чтобы понять, как это работает, предположим, что вы создаете систему SwimStats, а команда готова к реализации такой истории: «Как тренер я могу управлять пловцами в моей команде». Команда разработчиков обсуждает историю с тренерами/пользователями и выясняет, что тренер под этим понимает возможность добавлять новых пловцов, редактировать существующие данные по пловцам и удалять записи о пловцах, которые ушли из команды. Подобная первоначальная история легко разбивается на три более мелкие истории:

- Как тренер я могу добавлять новых пловцов в мою команду.
- Как тренер я могу редактировать информацию о пловцах, входящих в мою команду.
- Как тренер я могу удалять записи о пловцах, которые больше не входят в мою команду.

Эти истории практически совпадают с операциями создания, редактирования и удаления. Разбивка большой истории на эти три более мелкие истории представляет очень распространенную модель и дает нам следующее правило:

Разбивайте большие истории на отдельные CRUD-операции.

Удаление сквозной функциональности

В типичном приложении всегда есть множество ортогональных, или сквозных, функций. Безопасность, обработка ошибок и регистрация, например, пересекаются со всеми другими функциями приложения. Слишком большую историю, которая не укладывается в одну итерацию, нередко можно сократить путем ее изоляции от одной или нескольких сквозных функций.

Так, многие приложения содержат окна, которые ведут себя по-разному в зависимости от привилегий текущего пользователя. Если это слишком сложно реализовать за одну итерацию, разработайте окно за первую итерацию и добавьте поддержку привилегий конкретных пользователей в последующей итерации.

В одном из моих проектов клиенту нужно было разбить историю, которая предполагала поиск данных и отображение результатов. Поиск каждый раз проводился по всей базе данных, однако отображаться должны были только те результаты, к которым пользователь имел право доступа. Команда решила игнорировать это ограничение, связанное с безопасностью, и позволить пользователям видеть весь набор результатов после первой итерации.

В качестве другого примера предположим, что команда планирует работу над историей, в которой говорится: «Как пользователь я должен зарегистрироваться с использованием своих имени и пароля, чтобы войти в систему». Команда обсуждает эту историю и предлагает перечень ограничений на пароли: они должны иметь не менее восьми символов, содержать как минимум одну цифру, не должны содержать три и более повтора одного символа, должны шифроваться при передаче и т.д.

Ни одно из этих требований не является слишком трудоемким, однако в сумме они могут сделать историю слишком большой, чтобы она уложилась в одну итерацию. Проблему можно устранить путем удаления мер безопасности из истории, предусматривающей регистрацию, и создания незащищенной и защищенной версий истории. Вторая история может содержать перечень запланированных мер безопасности, таких как длина пароля, ограничения на использование символов и пр. Возможно, вы не решитесь на выпуск релиза только с первой, незащищенной версией истории, однако разбивка первоначальной, большой истории таким образом создает определенную стоимость.

Это подводит нас к следующему правилу:

Попробуйте удалить сквозные функции (такие как безопасность, регистрация и обработка ошибок) и создать две версии истории: одну

с поддержкой сквозных функций, а другую без поддержки.

Несоблюдение требований к быстродействию

При разработке программного обеспечения мы нередко забываем (или игнорируем) совет Кернигана и Пلودжера: «Сделайте это работоспособным, а затем повышайте быстродействие» (Kernighan and Plauger, 1974). Несколько лет назад я работал над проектом по воспроизведению графиков цен фондового рынка. Веб-пользователи должны были запрашивать график по биржевому символу (тикеру) акции. Наша программа должна была затем загружать исторические цены этой акции (за разные периоды от текущего дня до последних пяти лет) и строить линейный график движения акции. В число условий удовлетворенности, связанных с этой функцией, входили точность линии, восполнение пропусков данных и быстродействие. Для достижения целевого быстродействия мы должны были кэшировать наиболее часто запрашиваемые графики и перезаписывать каждый из них раз в минуту. Поскольку кэширование составляло значительную часть трудозатрат на разработку этой новой функции, мы выделили его в новую пользовательскую историю и включили в план следующей итерации. В общем смысле этот подход можно применять к любому нефункциональному требованию, в результате мы получаем следующее правило:

Попробуйте разбивать большие истории, разделяя функциональные и нефункциональные аспекты на отдельные истории.

Например, этот подход можно применить, если новая функция должна занимать менее установленного объема памяти, выстраивать линию, имеющую меньше определенного числа форм, или занимать критические ресурсы менее определенного количества времени.

Разбивка историй со смешанным приоритетом

В некоторых случаях одна история содержит несколько более мелких субисторий с различной приоритетностью. Допустим, проект включает в себя типичную историю, связанную с регистрацией: «Как пользователь я должен зарегистрироваться, чтобы войти в систему». Владелец продукта обозначает свои условия удовлетворенности для этой истории, которые

включают в себя следующее:

- Если пользователь вводит корректные имя и пароль, он получает доступ.
- Если пользователь вводит некорректный пароль три раза подряд, он лишается доступа до тех пор, пока не свяжется со службой поддержки клиентов.
- Если пользователю отказали в доступе, на его адрес направляется электронное письмо о том, что была совершена попытка использовать его учетные данные для получения доступа.

Эта история слишком велика для реализации в одну итерацию, поэтому ее необходимо разбить. Разбивку можно осуществить путем выявления низкоприоритетных элементов. В нашем случае владелец продукта не может отгрузить продукт, если он не поддерживает ключевую функциональность, связанную с регистрацией. Он может, однако, выпустить продукт без механизма ограничения времени на повторный ввод данных или без отправки электронного письма о попытке получения доступа. Это позволяет нам сформулировать еще одно правило разбивки историй:

Разбивайте большую историю на более мелкие, если маленькие истории имеют разные приоритеты.

Не разбивайте историю на задачи

Иногда попадают функции, которые сложно разделить. В таких ситуациях возникает соблазн разбить их на составляющие задачи. Для большинства разработчиков программного обеспечения анализ функции и разбивка ее на составляющие задачи настолько привычны, что выполняются практически автоматически. Вместе с тем избегайте разбивки истории следующим образом:

- Программирование пользовательского интерфейса.
- Написание среднего яруса.

Избежать этого соблазна лучше всего помогает рекомендация Ханта и Томаса (Hunt and Thomas, 1999) применить к системе так называемый подход «трейсер буллет» (tracer bullet). Трейсер буллет затрагивает все слои функции. Это может означать поставку частично реализованного пользовательского интерфейса, среднего яруса и базы данных. Поставка

связного подмножества всех слоев функции почти всегда лучше поставки одного полного слоя. Это дает нам следующее правило:

Не разбивайте большую историю на задачи. Вместо этого постарайтесь применить ко всей истории подход «трейсер буллет».

Избегайте соблазна добавить взаимосвязанные изменения

После разбивки истории и доведения ее до подходящего размера не усугубляйте ситуацию, добавляя работу в историю. Нередко возникает соблазн добавить так называемые взаимосвязанные изменения. Мы говорим себе: «Раз уж я работаю с этой программой, почему бы не внести в нее другие давно напрашивающиеся изменения». Под ними может подразумеваться исправление обнаруженных ошибок или решение какой-либо старой проблемы попутно с работой над отдельной задачей в этой же части программы. Вместе с тем приоритетность таких действий должна определяться точно так же, как приоритетность других функций. Другими словами, что имеет более высокий приоритет — растянувшееся на полдня устранение ошибки годичной давности или затрата такого же количества времени на что-нибудь другое? Ответ, понятно, полностью зависит от контекста. Таким образом, мы получаем последнее правило этой главы:

Избегайте ухудшения ситуации в результате добавления взаимосвязанных изменений к соответствующим образом оцененной функции за исключением случаев, когда эти взаимосвязанные изменения имеют такой же приоритет.

Объединение историй

После всех этих рекомендаций по разбивке историй может показаться, что все пользовательские истории, с которыми предстоит работать, должны быть как можно меньше. Это не так. Для команд, работающих двухнедельными итерациями, приемлема разбивка историй на такие части, на реализацию которых требуется примерно от двух до пяти дней. (Истории можно по-прежнему оценивать в пунктах, однако к тому моменту, когда

команда доберется до разбивки, она уже будет знать, сколько пунктов или идеальных дней эквивалентно сроку два–пять дней.) Истории должны быть немного меньше однонедельной итерации и могут быть, но необязательно, немного крупнее для более продолжительных итераций. Истории с такими приблизительными размерами лучше всего укладываются в короткие итерации agile-проектов.

Наряду с разбивкой больших историй может потребоваться и объединение нескольких крошечных историй. Объединенные истории оцениваются как единое целое, а не индивидуально. При возможности старайтесь объединять взаимосвязанные истории, поскольку в этом случае их легче приоритизировать. Очень распространена практика объединения нескольких отчетов об ошибках и работа с ними как с одним объектом.

Резюме

Иногда очень полезно разбить на части историю, которая не укладывается в одну итерацию из-за того, что она либо слишком велика для целой итерации, либо велика для оставшегося в планируемой итерации времени. Также полезно разбивать большие истории, когда требуется более точная оценка, чем та, которую можно получить для целой истории.

Историю можно разбивать по типу данных, которые она поддерживает. Ее также можно разбивать на основе предусматриваемых операций. Распространенной практикой является подход к разбивке на основе CRUD-операций (создание, чтение, редактирование и удаление). Историю можно уменьшить с помощью выделения сквозных функций, таких как безопасность, регистрация и обработка ошибок. Также это можно сделать через игнорирование целевой производительности во время той итерации, в которой реализуется функционал истории. Целевую производительность можно выделить в самостоятельную историю и реализовать позднее. Многие истории описывают две и более потребности. Если такие потребности имеют разный приоритет, разбивайте истории по ним.

Избегайте разбивки историй на задачи разработки, которые обязательны для реализации функции. Разбивка работы на обязательные задачи настолько привычна для нас, что мы начинаем автоматически разбивать на них пользовательские истории. Избегайте увеличения размера большой истории в результате включения в нее связанных изменений, которые не требуются для реализации этой пользовательской истории.

Наконец, помните, что иногда полезно объединять пользовательские истории, особенно в случае устранения ошибок, которые бывают слишком мелкими сами по себе.

Вопросы для обсуждения

1. Есть ли в вашем текущем или другом проекте пользовательские истории, которые трудно поддаются разбивке? Как бы вы подошли к их разбивке теперь?
2. К каким проблемам, на ваш взгляд, приводит разбивка истории на задачи с последующей придачей этим задачам статуса пользовательских историй?

Часть IV

Составление календарных графиков

В двух предыдущих частях рассматривалась оценка размера каждой желательной новой функциональности, а затем приоритизация, позволяющая ориентировать команду на создание наилучшего продукта. В настоящей части книги мы остановимся на составлении календарных графиков.

Следующие две главы начинаются с рассмотрения принципиальных этапов планирования релиза, а затем итерации. За ними идет глава, в которой предлагаются рекомендации по выбору оптимальной длины итерации. Помимо определения размера подлежащей выполнению работы, если мы хотим узнать, сколько времени займет проект, нам необходимо оценить скорость выполнения этой работы командой. Таким образом, темой очередной главы является оценка скорости.

Две последние главы этой части посвящены разбору ситуаций с высокой сложностью или неопределенностью. В первой из них мы будем говорить о том, как планировать проект, когда существует высокая вероятность ошибок в календарном графике или когда предварительный календарный график приходится составлять с большим упреждением на основе очень ограниченной информации. Настоящая часть завершается главой с описанием дополнительных аспектов планирования проектов с участием нескольких команд.

Глава 13

Основные аспекты планирования релиза

Ты импровизируешь. Ты адаптируешься. Ты преодолеваешь.

Клинт Иствуд в фильме «Перевал разбитых сердец»

Планирование релиза — это процесс создания высокоуровневого плана, который охватывает более длинный период, чем одна итерация. Разработка типичного релиза занимает от трех до шести месяцев и предполагает от трех до 12 и более итераций в зависимости от продолжительности итерации. План релиза важен по ряду причин.

Во-первых, он помогает владельцу продукта и команде в целом понять, сколько всего необходимо разработать и сколько времени потребуется, чтобы получить пригодный для поставки продукт. Чем быстрее продукт будет выпущен (и чем лучше этот продукт будет), тем скорее организация начнет получать отдачу от своих вложений в проект.

Во-вторых, план релиза выражает ожидания относительно того, что должно быть разработано и в какие сроки. Многим организациям эта информация необходима по той причине, что она подпитывает стратегическое планирование в целом.

В-третьих, план релиза служит ориентиром, по отношению к которому команда может оценивать прогресс. Без концепции релиза команды бесконечно переходят от одной итерации к другой. План релиза создает контекст, который позволяет объединить итерации в отвечающее запросам потребителей целое. Это фундаментальная цель любого итеративного процесса, а не только agile-процесса. Турист, который хочет добраться до вершины горы, может ориентироваться на самый высокий видимый пик. Однако после достижения этой вершины он узнает, что пик был ложным. Более высокую вершину не было видно за тем пиком, который он покорил. Турист нацеливается на эту, более высокую вершину, но лишь обнаруживает, что и она является ложной, поскольку теперь ему видна еще более высокая вершина. Такой не прямой путь покорения вершин в равной мере применим и к итеративным проектам. К счастью, проблемы, которые

он создает, устраняются в результате составления плана релиза, показывающего текущие ожидания команды в отношении того, куда она хочет в конечном итоге добраться.

План релиза

Частью планирования релиза является определение, какой объем работы можно выполнить и к какой дате. В одних случаях мы начинаем с даты и смотрим, какой объем работы можно завершить. В других случаях мы начинаем с набора пользовательских историй и смотрим, сколько времени потребуется для их реализации. В обоих случаях, как только команда получает первоначальный ответ, он оценивается относительно целей организации в проекте: принесет ли разработанный продукт желаемое количество денег? Поможет ли он сэкономить достаточно денег? Позволит ли продукт получить целевую долю рынка? Если нет, то, может быть, более продолжительный или короткий проект позволит достичь приемлемой комбинации целей.

В первом приближении определение того, какой объем работы потребуется для релиза и какие пользовательские истории войдут в него, выглядит очень простым — умножение планового количества итераций на ожидаемую или известную скорость команды дает нам общий объем работы, которую можно выполнить. Затем подбирается такое количество пользовательских историй, которое соответствует общему объему работы. Предположим, что мы хотим отгрузить новый продукт через шесть месяцев. Мы планируем работать двухнедельными итерациями, поэтому в нашем проекте будет 13 итераций. Мы ожидаем, что скорость команды составит 20 пунктов или идеальных дней на итерацию. Тогда размер всего проекта будет равен $13 \times 20 = 260$ пунктов или идеальных дней. Теперь владелец продукта и команда могут обсудить все истории и определить их приоритет, с тем чтобы создать наибольшую стоимость и при этом не выйти за пределы 260. План релиза обычно представляет собой перечень пользовательских историй, которые будут реализованы в течение проекта.

Планирование релиза — это не создание плана, указывающего, какие разработчики будут заниматься теми или иными пользовательскими историями или задачами, или последовательность выполнения работ в пределах одной итерации. Заниматься составлением плана с таким уровнем детализации во время планирования релиза опасно и может вводить в заблуждение. Решения о том, кто и чем будет заниматься, и о последовательности работ лучше всего оставить тем, кто будет выполнять эти задачи, и отодвинуть по срокам как можно дальше. Кроме того,

помните, что объектами в плане релиза являются пользовательские истории, которые представляют собой описание предоставляемой функциональности, а не конкретных технических задач, подлежащих выполнению. Планирование релиза — слишком ранний этап проекта, чтобы в достаточной мере понимать пользовательские истории и разбивать их на технические задачи. Как вы увидите в главе 14 «Планирование итерации», команда в конечном итоге разбивает пользовательские истории из плана релиза на составляющие их задачи. Однако она не делает этого до начала итерации, содержащей эти истории.

Конечно, если ваш проект, организация и условия работы требуют этого, вы можете включить в план релиза дополнительную информацию. Например, вы можете изложить ключевые допущения, лежащие в основе плана. Прежде всего это касается допущений в отношении того, кто войдет в состав команды, какой будет продолжительность итерации, даты начала первой итерации и даты завершения последней итерации. В главе 21 «Информирование о плане» приводится дополнительная полезная информация, которую можно использовать при доведении плана релиза до окружающих.

Основные этапы планирования релиза показаны на рис. 13.1. Каждый из этих этапов описывается в следующих разделах.

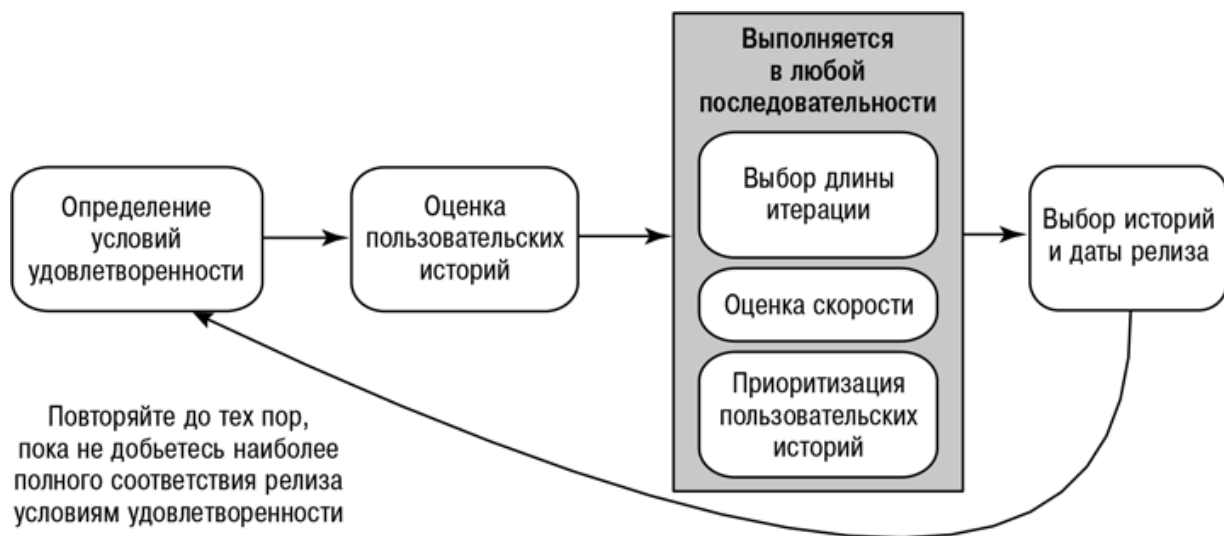


Рис. 13.1. Этапы планирования релиза

Определение условий удовлетворенности

Прежде чем приступить к планированию релиза, важно установить критерии, по которым можно судить о том, насколько успешен или неудачен проект. Для большинства проектов конечным критерием является то, сколько денег они позволили сэкономить или заработать. В качестве основных индикаторов вероятности достижения финансовых целей в

большинстве проектов используются календарный график, объем работы и ресурсы. Это означает в большинстве случаев, что условия удовлетворенности владельца продукта определяются сочетанием целевых сроков, объема и ресурсов.

Владелец продукта обозначает желаемые целевые уровни для каждого из этих факторов почти на каждом совещании по планированию релизов. Владелец продукта, например, может хотеть, чтобы за три месяца были реализованы четыре темы (оцененные в 200 пунктов) без добавления персонала. Хотя цель для каждого фактора нередко требует определения, один фактор обычно хорошо известен. Проекты бывают ориентированными либо на сроки, либо на функции. *Ориентированным на сроки* считается такой проект, который должен быть завершен к определенной дате, при этом набор его функций определяется по договоренности. *Ориентированным на функции* считается такой проект, который должен быть завершен в возможно короткие сроки, но при этом реализация набора функций считается более важной.

Оценка пользовательских историй

Поскольку оценка отражает затраты на реализацию пользовательской истории, очень важно оценить каждую из них. Представьте, что вы решили заменить все предметы одежды в своем гардеробе. Вы приходите в магазин, и тут обнаруживается, что ценники на одежде отсутствуют и вы не можете узнать, что сколько стоит. Именно в такой ситуации оказывается владелец продукта, которому не предоставляют никаких оценок.

Не обязательно оценивать подряд все, что захочется владельцу продукта. Оценки требуют только те новые функции, которые с разумной вероятностью попадут в ближайший релиз. Нередко владелец продукта имеет список пожеланий, который включает в себя два, три или более будущих релиза. Оценивать более отдаленную работу не стоит.

Выбор длины итерации

Большинство agile-команд работают с итерациями продолжительностью от двух до четырех недель. Итерации могут быть и чуть длиннее, а некоторые команды экспериментируют с более короткими сроками. При планировании релиза необходимо выбрать подходящую длину итерации. Рекомендации по выбору представлены в главе 15 «Выбор длины итерации».

Оценка скорости

Если команда имеет опыт совместной работы, зачастую лучше всего

использовать ее скорость в последнем проекте. Естественно, в случае кардинального изменения технологии или сферы бизнеса прошлая скорость команды может оказаться непригодной для использования. Вместе с тем существуют методы, позволяющие делать обоснованные оценки скорости, опираясь на прошлые результаты. В главе 16 «Оценка скорости» мы рассмотрим эти методы, а также возможные варианты оценки скорости.

Приоритизация пользовательских историй

В большинстве проектов либо слишком мало времени, либо слишком много функций. Зачастую невозможно выполнить все, что хочет каждый, за отведенное время. Из-за этого владельцу продукта приходится устанавливать приоритеты для функций, которые он хочет получить. Хороший владелец продукта берет на себя абсолютную ответственность за приоритизацию, однако прислушивается к рекомендациям команды разработчиков, особенно в отношении последовательности реализации. Пользовательские истории приоритизируются на основе правил, рассмотренных в предыдущей части настоящей книги.

Выбор историй и даты релиза

К этому моменту у вас уже есть оценка скорости команды на итерацию и предположение в отношении количества необходимых итераций. Теперь пришло время посмотреть, можно ли спланировать такой релиз, который соответствует условиям удовлетворенности для проекта.

Если проект ориентирован на функции, мы можем просуммировать оценки всех необходимых функций и разделить сумму на ожидаемую скорость. Это даст нам количество итераций, необходимых для реализации желаемой функциональности.

Если проект ориентирован на сроки, мы можем определить количество итераций, глядя на календарь. Умножение количества итераций на ожидаемую скорость покажет нам, сколько пунктов или идеальных дней уложатся в релиз. Это количество пунктов или идеальных дней можно соотнести с приоритизированным перечнем пользовательских историй и понять, какую функциональность возможно реализовать к желаемому сроку.

Следующий вопрос касается того, насколько детальным будет план релиза. Одни команды в определенных условиях предпочитают создавать планы релизов, которые показывают, что они предполагают разработать в каждой итерации. Другие команды предпочитают просто определять, что должно быть реализовано в релизе в целом, оставляя проработку конкретики каждой итерации на более позднее время. В процессе

планирования релиза команды должны обсудить этот вопрос и принять соответствующее решение.

У каждого подхода есть свои преимущества и недостатки. Понятно, что распределение конкретных функций по конкретным итерациям занимает больше времени. Вместе с тем прорабатываемые при этом детали могут быть полезными при координировании работы нескольких команд. Отказ от распределения работы по конкретным итерациям обеспечивает меньшую детализацию, но отнимает намного меньше времени. Помимо прочего, в случае предварительного распределения работы по конкретным итерациям мы делаем это на основе меньшего объема знаний, чем при распределении перед началом каждой итерации. План, несомненно, будет изменяться по мере того, как мы узнаем больше в ходе выполнения проекта. Как следствие, затраты времени и сил на распределение работы по конкретным итерациям необходимо взвешивать относительно этого аспекта. Так или иначе, в некоторых проектах данный подход имеет смысл. Разумным компромиссом, на мой взгляд, является распределение конкретных работ по первым двум-трем итерациям и представление оставшейся части плана релиза в виде единого блока. Практически всегда стоит определять конкретные работы для первой итерации, особенно если она начинается сразу же.

Из-за множества мнений и вопросов типа «что... если...» во время типичной сессии по планированию релиза неплохо было бы иметь удобный способ управления входными и выходными параметрами релиза и итерациями. Лучше всего, при условии что каждый знает свою задачу, работать с карточками размером 6×12 см или стикерами с пользовательскими историями или функциями. В отличие от программных средств карточки осязаемы, и их легко может прочесть каждый.

Для планирования релиза владелец продукта выбирает наиболее приоритетные объекты, которые укладываются в первую итерацию. Карточки располагают стопками или распределяют так, чтобы было понятно, какие истории входят в какую итерацию. На рис. 13.2 показан один из подходов к выполнению этой задачи. Карточки можно разложить на столе или развесить на стене. Размещать карточки удобнее всего на пробковой доске для записок, которая позволяет прикреплять карточки кнопками, а не приклеивать их. Поскольку на каждой карточке обозначена оценка истории, можно быстро пробежать по колонкам, показанным на рис. 13.2, и удостовериться, что в каждой итерации обеспечен необходимый объем работы.

1	2	Итерации 3–5		
Как пользователь я... 3	Как пользователь я... 6	Как пользователь я... 5	Как пользователь я... 4	Как пользователь я... 3
Как пользователь я... 5	Как пользователь я... 5	Как пользователь я... 5	Как пользователь я... 5	Как пользователь я... 5
Как пользователь я... 3	Как пользователь я... 2	Как пользователь я... 1	Как пользователь я... 4	Как пользователь я... 4
Как пользователь я... 2		Как пользователь я... 2		Как пользователь я... 1

Рис. 13.2. Размещение итераций в виде колонок

Обновление плана релиза

Итак, план релиза составлен. Вместе с тем очень важно, чтобы он не превращался в нечто незыблемое, не сдавался в архив и не пылился на полке. План релиза необходимо регулярно пересматривать и обновлять. Если скорость команды разработчиков остается довольно стабильной, а планирование итераций не приносит больших сюрпризов, то можно работать от четырех до шести недель без официального обновления плана релиза. Однако во многих проектах полезно взять за правило пересматривать план релиза после каждой итерации.

Пример

Чтобы связать все сказанное вместе, рассмотрим небольшой пример. Обратимся к уже знакомому нам сайту SwimStats. В результате исследования рынка мы установили, что тренеры и пловцы в нашей целевой аудитории хотят видеть функции, описываемые следующими пользовательскими историями:

- Как тренер я могу определять расписание тренировок.
- Как тренер я могу определять по своему желанию поля для отслеживания по каждому пловцу.
- Как пловец я могу получить отчет, который показывает рост моих

результатов по всем соревнованиям с определенной прошлой даты.

- Как пловец я могу корректировать свою гендерно-возрастную информацию.
- Как тренер я могу рассылать электронные письма всем пловцам команды.
- Как системный администратор я могу конфигурировать права доступа и пользовательские группы.
- Как тренер я могу приобретать программу SwimStats и использовать ее вместе со своей командой.
- Как пловец я могу сравнивать свои результаты с национальными рекордами.
- Как тренер я могу вводить имена и гендерно-возрастную информацию по всем пловцам моей команды.
- Как пловец я могу видеть мое лучшее время в каждом соревновании.
- Как пользователь я должен зарегистрироваться в системе и могу видеть только данные, к которым мне разрешен доступ.
- Как пловец я могу видеть все мои результаты в конкретных соревнованиях.

Определение условий удовлетворенности

Весенний плавательный сезон начинается через четыре месяца. Первый релиз SwimStats должен быть доступным за месяц до этого. Владелец продукта хочет получить как можно больше функций, однако это проект, ориентированный на сроки. Что-то обязательно надо выпустить через три месяца, даже если это будут лишь основные аспекты для небольших команд. К тому же компания, как стартап, имеет фиксированный бюджет. Проект необходимо выполнить силами одного программиста, одного администратора баз данных и одного тестировщика, которые уже находятся в штате.

Оценка

Три разработчика, которые будут заниматься проектом, встречаются с владельцем продукта. Они задают вопросы по пользовательским историям, а владелец продукта поясняет цель каждой из них. Команда принимает решение использовать для оценки пункты и присваивает историям значения, показанные в табл. 13.1.

Таблица 13.1. Приоритизированный перечень историй

История	Оценка
Как тренер я могу приобретать программу SwimStats и использовать ее вместе со своей командой	5
Как пользователь я должен зарегистрироваться в системе и могу видеть только данные, к которым мне разрешен доступ	5
Как тренер я могу вводить имена и гендерно-возрастную информацию по всем пловцам моей команды	8
Как пловец я могу видеть все мои результаты в конкретных соревнованиях	5
Как пловец я могу корректировать свою гендерно-возрастную информацию	5
Как пловец я могу видеть мое лучшее время в каждом соревновании	5
Как пловец я могу получить отчет, который показывает рост моих результатов по всем соревнованиям с определенной прошлой даты	5
Как пловец я могу сравнивать свои результаты с национальными рекордами	8
Как тренер я могу определять расписание тренировок	8
Как тренер я могу рассылать электронные письма всем пловцам команды	3
Как тренер я могу определять по своему желанию поля для отслеживания по каждому пловцу	8
Как системный администратор я могу конфигурировать права доступа и пользовательские группы	3
Всего	68

Выбор длины итерации

Поскольку на весь проект отведено только три месяца, разработчики и владелец продукта соглашаются с тем, что четырехнедельные итерации слишком велики. Они не дают достаточного количества точек контроля прогресса команды или не оставляют владельцу продукта достаточных возможностей для управления проектом путем корректировки приоритетов. На основе аспектов, которые будут представлены в главе 15 «Выбор длины итерации», все согласились на двухнедельные итерации.

Оценка скорости

В этом проекте будут заняты два программиста и один тестировщик. Используя приемы, описанные в главе 16 «Оценка скорости», они закладывают скорость восемь пунктов на итерацию.

Приоритизация пользовательских историй

Владелец продукта, опираясь на выполненное исследование рынка, расставляет приоритеты историй. Перечень приоритизированных историй вместе с оценкой каждой из них приведен в табл. 13.1.

Выбор пользовательских историй

Поскольку это проект, ориентированный на сроки, мы знаем, сколько итераций в нем может быть. Владелец продукта хочет получить релиз через три месяца — это шесть двухнедельных итераций плюс дополнительная неделя для непредвиденных случаев. При предполагаемой скорости восемь пунктов на итерацию релиз будет включать $6 \times 8 = 48$ пунктов. Теперь владелец продукта может выбрать для включения в релиз работу объемом до 48 пунктов из табл. 13.1.

Пользовательские истории в табл. 13.1 рассортированы владельцем продукта по убыванию их ценности для первого релиза SwimStats. Владелец продукта может выбрать до 48 пунктов. Первоначально он выбирает первые восемь историй, включая «Как пловец я могу сравнивать свои результаты с национальными рекордами». Это составляет в сумме 46 пунктов, что довольно близко к согласованным предельным 48 пунктам.

Разработчики, однако, указывают на необходимость включения истории, получившей у владельца продукта низший приоритет: «Как системный администратор я могу конфигурировать права доступа и пользовательские группы». Эта история оценивается в три пункта, что увеличивает объем работы в релизе до 49 пунктов при плановых 48. Плановый объем в 48 пунктов — величина довольно приблизительная, а кроме того, есть еще дополнительная неделя после шестой итерации. При таких обстоятельствах вполне возможно пойти на увеличение объема до 49 пунктов.

Так или иначе, команда SwimStats не решается на увеличение объема до 49 пунктов. Если добавить трехпунктовую историю, то из релиза необходимо исключить как минимум один пункт. Владелец продукта принимает решение удалить историю «Как пловец я могу сравнивать свои результаты с национальными рекордами» на восемь пунктов. Это уменьшает релиз до 41 пункта. Владелец продукта мог бы добавить трехпунктовую историю «Как тренер я могу рассылать электронные письма всем пловцам команды», но он предпочитает воздержаться от этого. Если команда выполнит хотя бы один пункт раньше календарного графика, то он предпочтет вернуть назад исключенную восьмипунктовую историю. В результате окончательный план релиза принимает форму, приведенную в табл. 13.2.

Таблица 13.2. Окончательный план релиза для SwimStats

История	Оценка
Как тренер я могу приобретать программу SwimStats и использовать ее вместе со своей командой	5
Как системный администратор я могу конфигурировать права доступа и пользовательские группы	3
Как пользователь я должен зарегистрироваться в системе и могу видеть только данные, к которым мне разрешен доступ	5
Как тренер я могу вводить имена и гендерно-возрастную информацию по всем пловцам моей команды	8
Как пловец я могу видеть все мои результаты в конкретных соревнованиях	5
Как пловец я могу корректировать свою гендерно-возрастную информацию	5
Как пловец я могу видеть мое лучшее время в каждом соревновании	5
Как пловец я могу получить отчет, который показывает рост моих результатов по всем соревнованиям с определенной даты в прошлом	5
Всего	41

Резюме

План релиза — это план высокого уровня, который охватывает более продолжительный период, чем одна итерация. Большинство команд выпускают релизы каждые три–шесть месяцев, однако нет ничего необычного в более частом или редком выпуске релизов в зависимости от типа программного обеспечения. На простейшем уровне планирование релиза тривиально: умножьте ожидаемую скорость на количество планируемых итераций, а затем выберите истории, сумма оценок которых соответствует релизу.

План релиза не должен в точности описывать, что́ будет разрабатываться в течение каждой итерации. В реальности такой уровень детализации требуется редко. Для большинства проектов вполне достаточно идентифицировать истории, которые пойдут в работу в первой паре итераций, а остальные истории должны распределяться по конкретным итерациям позднее.

Планирование релиза — итеративный процесс, который начинается с идентификации условий удовлетворенности владельца продукта. Первоначально к этим условиям относятся целевые календарный график, объем работы и ресурсы. Если проект нельзя спланировать так, чтобы он соответствовал набору первоначальных условий удовлетворенности, то процесс планирования повторяют и пытаются выполнить урезанный набор

условий. Не исключено, что желаемая функциональность может быть реализована немного позже или с привлечением более многочисленной команды.

После создания плана релиза его не оставляют висеть на стене без изменений, а обычно обновляют перед началом каждой итерации.

Вопросы для обсуждения

1. Каким является ваш текущий проект, ориентированным на сроки или на функции? Почему он именно такой?
2. Как часто в вашей организации или в вашем текущем проекте должен обновляться план релиза?

Глава 14

Планирование итерации

Это серьезнейшая ошибка, строить теории до того, как получишь информацию.

Шерлок Холмс в «Скандале в Богемии» Артура Конана Дойла

План релиза — это превосходное представление высокого уровня о том, как команда собирается создавать предельно ценный продукт. Вместе с тем план релиза дает лишь очень крупную картину процесса создания продукта. Он не позволяет увидеть более краткосрочные, более детальные ориентиры, которые необходимы для управления работой в пределах отдельной итерации. Составив план итерации, команда получает более сфокусированное, детальное представление о том, что необходимо для полной реализации только тех пользовательских историй, которые отобраны для новой итерации.

План итерации — это результат совещания, посвященного планированию итерации. На этом совещании должны присутствовать владелец продукта, аналитики, программисты, тестировщики, администраторы баз данных, дизайнеры пользовательских интерфейсов и т.д. В общем, присутствовать необходимо всем, кто имеет отношение к осмыслению исходной идеи и превращению ее в работоспособный продукт.

В реальности план итерации может представлять собой простую электронную таблицу или комплект карточек с написанными на них задачами. В любом случае задачи и истории необходимо организовать так, чтобы можно было понять, к каким историям какие задачи относятся. Например, посмотрите на табл. 14.1, где показан план итерации в электронной таблице. Задачи расположены с отступом по одной в строке под историей, к которой они относятся.

Альтернатива электронной таблице представлена на рис. 14.1, который иллюстрирует использование карточек для планирования итерации. Карточки можно расположить, как на этом рисунке, на столе или на полу, а также приклеить их или прикрепить кнопками к стене. Лично я предпочитаю планировать итерации с использованием карточек. Члены команды могут покидать совещание по планированию и сразу вводить

карточки в компьютерную систему, если им хочется, но на самом совещании намного удобнее пользоваться карточками.

Таблица 14.1. План итерации, представленный в виде простой электронной таблицы

Строка	Пользовательская история/задача	Часы
1	Как тренер я могу назначать пловцов на заплывы в соревнованиях	
2	Определение правил по выбору тех, кто и в каких заплывах может выступать	6
3	Определение приемочных тестов для демонстрации того, как это должно работать	8
4	Дизайн пользовательского интерфейса	16
5	Кодирование пользовательского интерфейса	8
6	Добавление таблиц и хранимых процедур в базу данных	6
7	Автоматизация тестирования	6
8	Как пловец я могу корректировать свою гендерно-возрастную информацию	
9	Определение приемочных тестов	5
10	Изменение страницы гендерно-возрастной информации, представленной в режиме просмотра, так, чтобы сделать возможным редактирование	6
11	...	

Одним из самых значительных преимуществ использования карточек во время планирования итерации является то, что они позволяют участвовать в процессе всем. Если задачи вводятся в компьютерную систему во время совещания по планированию итерации, с клавиатурой работает кто-то один. Контроль над клавиатурой дает огромную власть. Наборщик должен участвовать во всех разговорах, иначе ничего не попадет в план. Еще хуже то, что владелец клавиатуры может изменять все вводимое в план.

Два примера наглядно демонстрируют эту власть. В первом случае команда обсуждала конкретный элемент и решила оценить его в 12 часов. Клавиатура была в распоряжении человека, сочетавшего должности руководителя проекта/технического руководителя. Он ввел в систему оценку восемь, поскольку «этим невозможно заниматься так долго», даже несмотря на то, что он не был тем, кто будет выполнять эту задачу.

История	Задачи	
Как тренер я могу назначать пловцов на заплывы в соревнованиях	Определение правил по выбору тех, кто и в каких заплывах может выступать 6	Определение приемочных тестов для демонстрации того, как это должно работать 8
	Дизайн пользовательского интерфейса 16	Кодирование пользовательского интерфейса 8
Как пловец я могу корректировать свою гендерно-возрастную информацию	Определение приемочных тестов 5	Изменение страницы гендерно-возрастной информации, представленной в режиме просмотра, так, чтобы сделать возможным редактирование 6

Рис. 14.1. Планировать итерации можно с помощью карточек на столе или стене

Во втором случае команда, которую я обучал, обсуждала, как необходимо реализовывать новую функцию — будет ли это Java-код на сервере или хранимая процедура в базе данных? Все, кроме руководителя команды, который распоряжался клавиатурой, согласились с тем, что функция должна реализовываться с помощью хранимых процедур. Руководителя попросили создать задачу «Добавить хранимые процедуры» в электронную таблицу, а он вместо этого ввел «Написать код хранения данных». Его послание было ясным: эта проблема не решена.

Сравните эти две ситуации с совещанием по планированию итерации, на котором любой может взять карточку и написать на ней задачу в любой момент. Использование карточек — значительно более демократичный и коллективный подход и скорее принесет более высокие результаты в ходе итерации и проекта, а не только совещания.

Задачи, не распределенные во время планирования итерации

Прежде чем говорить о том, что происходит во время планирования итерации, необходимо прояснить еще один вопрос. В процессе

планирования итерации задачи не распределяются между конкретными исполнителями. В начале итерации и так очевидно, кто какой задачей будет заниматься. Вместе с тем по мере реализации командой полного набора задач то, что казалось очевидным вначале, не обязательно происходит в ходе итерации. Например, при планировании итерации мы можем предполагать, что наш администратор баз данных будет заниматься задачей «отладка расширенного поискового запроса», поскольку у нее самый большой опыт работы с языком SQL в команде. Впрочем, если у нее не будет возможности выполнить эту задачу, ею займется кто-нибудь другой.

Индивидуальные исполнители не распределяются по задачам до начала итерации и обычно получают всего одну-две связанные задачи за раз. Выполнение новых задач не начинается до тех пор, пока не будут реализованы задачи, выбранные ранее.

Распределение задач между конкретными исполнителями во время планирования итерации не дает ни плюсов, ни минусов для процесса. Проекты реализуются лучше всего, когда господствует идеология «мы все работаем над этим вместе», когда члены команды подчищают хвосты друг за другом, зная, что это окупится. Если задачи распределяются между конкретными исполнителями в начале итерации, это мешает общему стремлению к достижению целей итерации.

Чем различаются планирование итерации и планирование релиза

План релиза описывает процесс выпуска продукта, который обычно занимает от трех до шести месяцев с начала нового проекта. В отличие от этого план итерации охватывает только одну итерацию, продолжительность которой обычно составляет от двух до четырех недель. Пользовательские истории из плана релиза разбивают на задачи для плана итерации. Если пользовательские истории в плане релиза оцениваются в пунктах или идеальных днях, то задачи в плане итерации оцениваются в идеальных часах.

Почему задачи в плане итерации оцениваются в часах, а истории в плане релиза — в пунктах или идеальных днях? Прежде всего потому, что это осуществимо. Работа в итерации занимает не более нескольких недель, и команде необходимо иметь разумный уровень ее детализации, особенно после обсуждения на совещании по планированию итерации. Это позволяет ей достоверно оценивать задачи итерации в часах. Каждая из пользовательских историй, которые входят в релиз, представляет множество задач, более неопределенна и менее понятна, а поэтому требует оценки в

более абстрактных единицах, таких как пункты или идеальные дни.

Эти основные различия между планом релиза и планом итерации представлены в обобщенном виде в табл. 14.2.

Главная цель планирования итерации заключается в уточнении предположений, сделанных в более крупном плане релиза. План релиза обычно намеренно делают неопределенным в отношении конкретного порядка работы над пользовательскими историями. Кроме того, во время планирования итерации команда знает значительно больше, чем при последнем обновлении плана релиза. Планирование итерации позволяет команде опираться на самые последние знания. Как результат, agile-планирование превращается в двухступенчатый процесс. На первом этапе составляется план релиза с его шероховатостями и общей неопределенностью. На втором этапе разрабатывается план итерации, который по-прежнему имеет некоторые шероховатости и остается неопределенным. Вместе с тем, поскольку его составление совпадает с началом новой итерации, он более детализирован, чем план релиза.

Таблица 14.2. Основные различия между планом релиза и планом итерации

	План релиза	План итерации
Горизонт планирования	3–9 месяцев	1–4 недели
Элементы плана	Пользовательские истории	Задачи
Оцениваются	В пунктах или идеальных днях	В идеальных часах

Создание плана итерации подводит команду к обсуждению как дизайна продукта, так и дизайна программного обеспечения. Обсуждение дизайна продукта может, например, затрагивать такие вопросы, как наилучшая комбинация историй для оптимизации стоимости, интерпретация обратной связи после представления работающей программы клиентам и степень реализации желаемой функции (т.е. позволяет ли 20%-ная реализация функции продемонстрировать 80% потребительской стоимости). Обсуждение дизайна программного обеспечения, в свою очередь, может касаться выбора слоя архитектуры для реализации новой функции, выбора технологий и т.д. В результате такого обсуждения команда получает более глубокое представление о том, что должно быть создано и будет создаваться, а также составляет перечень задач, которые необходимо решить для достижения цели итерации.

Планирование итерации на основе скорости

В широком смысле существует два подхода к планированию итерации, которые я называю подходом *на основе скорости* и подходом *на основе обязательств*. Разные команды используют разные подходы, и каждый из них может быть успешным. Вдобавок эти общие подходы можно комбинировать в той или иной степени. В этом разделе мы рассмотрим планирование итерации на основе скорости, а в следующем сосредоточимся на планировании на основе обязательств.

Этапы планирования итерации на основе скорости представлены на рис. 14.2. Прежде всего команда коллективно корректирует приоритеты. В предыдущей итерации она может получить знания, заставляющие изменить приоритеты. Затем она определяет целевую скорость для предстоящей итерации. После этого команда выбирает цель итерации, которая представляет собой общее описание того, что она хочет получить в результате итерации. Выбрав цель итерации, команда переходит к выбору пользовательских историй с наивысшим приоритетом, которые поддерживают эту цель. Историй выбирают столько, чтобы сумма их оценок в пунктах или идеальных днях была равной целевой скорости. Наконец, каждую выбранную историю разбивают на задачи, а каждую задачу оценивают. Эти этапы описываются более подробно в оставшейся части этой главы.



Рис. 14.2. Последовательность этапов при планировании итерации на основе скорости

Корректировка приоритетов

Представьте, что все пользовательские истории физически сложены в стопку или рассортированы в электронной таблице так, что самая ценная из них находится наверху, а самая малоценная — внизу. По мере осуществления проекта перед началом каждой итерации всегда берутся верхние истории из этого приоритизированного перечня. Однако коммерческие и проектные условия меняются так быстро, что всегда есть основания для быстрого пересмотра приоритетов.

Один из источников изменения приоритетов — *совещание по анализу итерации*, которое проводится после завершения каждой итерации. В процессе анализа итерации новые функциональности и возможности, добавленные за эту итерацию, демонстрируются заинтересованным лицам, расширенному проектному сообществу и любым другим лицам. Анализ итерации нередко дает ценную обратную связь. Сам владелец продукта обычно не предлагает новые идеи или изменения во время анализа итерации, поскольку он и так повседневно участвовал в работе. Однако многие другие (включая потенциальных клиентов и пользователей) могут видеть результаты итерации в первый раз. Они нередко выдвигают хорошие новые идеи, которые могут вытеснять статьи, ранее имевшие высокий приоритет.

Как говорилось в главе 9 «Приоритизация тем», пользовательские истории и темы приоритизируются на основе их финансовой стоимости для продукта, затрат, количества и значимости новых знаний, полученных командой, и степени снижения риска. В идеале команда должна подождать до окончания совещания по анализу итерации, прежде чем обсуждать приоритеты для очередной итерации. В конце концов, то, что ее члены услышат в процессе анализа итерации, может повлиять на их мнение и позицию в отношении приоритетов для следующей итерации, если нет полной уверенности в содержании работ. По моему опыту, полезно проводить совещание по приоритизации за несколько дней до начала новой итерации. В этом случае легче назначить анализ итерации и совещание по планированию итерации на один и тот же день.

Анализ итерации обычно занимает от 30 до 60 минут. В случаях крупных продуктов при участии нескольких команд бывает, что владелец продукта и другие ключевые заинтересованные лица, необходимые для обсуждения приоритетов, тратят на анализ итерации полдня. Добавьте к этому четыре часа на планирование итерации, и у вас может не остаться времени для обсуждения приоритетов в тот же день.

Я обычно провожу совещание по приоритизации за два дня до конца итерации. К этому времени вполне ясно, останется ли незавершенная работа в текущей итерации. Это позволяет владельцу продукта решить, будет ли завершение такой работы приоритетным для очередной итерации. Владелец продукта проводит совещание по приоритизации и привлекает всех, кого считает необходимым, к участию в обсуждении приоритетов проекта. После этого совещания он может быстро и без усилий скорректировать приоритеты на основе результатов анализа итерации.

Определение целевой скорости

Следующий этап планирования итерации на основе скорости —

определение целевой скорости команды. По умолчанию большинство команд принимают свою скорость в следующей итерации равной скорости в ближайшей прошлой итерации. Бек и Фаулер (Beck and Fowler, 2000) называют это *вчерашней погодой*, поскольку самый лучший прогноз сегодняшней погоды тот, который похож на вчерашнюю погоду. Есть и такие команды, которые предпочитают использовать скользящие средние, скажем, за последние три итерации.

Если команда не работала вместе прежде или не знакома с agile-процессом, то ей придется спрогнозировать скорость. Методы прогнозирования скорости описаны в главе 16 «Оценка скорости».

Идентификация цели итерации

С учетом приоритетов и целевой скорости команда идентифицирует цель, которой она должна достичь во время итерации. Цель — короткое описание того, что она хотела бы реализовать за этот период. Например, команда SwimStats может выбрать в качестве цели итерации «Завершение всех гендерно-возрастных функций». Другие цели итераций для SwimStats могут включать в себя следующее:

- Добиться прогресса по отчетам.
- Завершить реализацию всех отчетов по результатам в соревновании.
- Обеспечить работоспособность функции безопасности.

Цель итерации — это единое заявление о том, что должно быть реализовано в течение данной итерации. Она не должна быть очень конкретной. Например, «Добиться прогресса по отчетам» — хорошая цель итерации. Ее не нужно делать более конкретной, например «Завершить 15 отчетов» или «Выполнить отчеты по результатам соревнования». Если «Добиться прогресса по отчетам» наглядно описывает, над чем придется работать в предстоящей итерации, то это хорошее заявление об этой цели.

Выбор пользовательских историй

На следующем этапе владелец продукта и команда выбирают истории, которые в совокупности обеспечивают достижение цели итерации. Если команда SwimStats выбирает в качестве цели итерации «Завершение всех гендерно-возрастных функций», она должна работать над любыми еще не реализованными историями, которые имеют отношение к гендерно-возрастным функциям. Они могут включать в себя:

- Как пловец я могу корректировать свою гендерно-возрастную информацию.

- Как тренер я могу вводить гендерно-возрастную информацию по всем пловцам моей команды.
- Как тренер я могу импортировать файл со всеми гендерно-возрастными данными.
- Как тренер я могу экспортировать файл со всеми гендерно-возрастными данными.

При выборе историй для работы владелец продукта и команда рассматривают приоритеты. Например, если экспорт файла с гендерно-возрастными данными находится внизу перечня приоритизированных требований к продукту, он может быть не включен в итерацию. В этом случае цель итерации лучше сформулировать следующим образом: «Завершение наиболее важных гендерно-возрастных функций».

Разбивка пользовательских историй на задачи

После выбора приемлемого набора пользовательских историй каждую из них разбивают на задачи, необходимые для создания новой функциональности. Допустим, наивысший приоритет имеет пользовательская история «Как тренер я могу расставлять пловцов по заплывам в предстоящих соревнованиях». Эта пользовательская история превращается в следующий перечень задач:

- Определение правил, которые влияют на то, кого можно поставить на какой заплыв.
- Написание тестовых сценариев, показывающих, как это должно работать.
- Разработка пользовательского интерфейса.
- Получение обратной связи по пользовательскому интерфейсу от тренеров.
- Кодирование пользовательского интерфейса.
- Кодирование среднего яруса.
- Добавление новых таблиц в базу данных.
- Автоматизация приемочных тестов.

Самый распространенный вопрос, связанный с планированием итерации, — что следует включать. Необходимо идентифицировать все задачи, требующиеся для перехода от пользовательской истории к функционирующему, завершенному продукту. При наличии задач, связанных с анализом, дизайном, разработкой пользовательского интерфейса и т.п., их необходимо идентифицировать и оценить. Поскольку целью каждой итерации является выпуск потенциально готового продукта, не забудьте включить задачи по тестированию и документированию.

Включение задач по тестированию важно по той причине, что команда должна уже в начале итерации определиться с тем, как будет тестироваться пользовательская история. Это позволяет задействовать тестировщиков прямо с начала итерации и, таким образом, поддержать кроссфункциональное поведение команды.

Включайте только ту работу, которая создает стоимость для соответствующего проекта

План итерации должен идентифицировать только те задачи, которые сразу создают стоимость для текущего проекта. Очевидно, что к ним относятся задачи, которые связаны с анализом, дизайном, кодированием, тестированием, разработкой пользовательского интерфейса и т.д. Не включайте час с утра, который тратится на ответы на электронные письма. Конечно, некоторые из этих писем связаны с проектом, но задачи вроде «ответы на электронные письма, один час» не следует включать в план итерации.

Точно так же следует поступать с обязательными совещаниями у директора компании по персоналу по новой процедуре ежегодной аттестации. Они не должны включаться в план итерации. Даже несмотря на то, что члены проектной команды будут проходить аттестацию по новой процедуре, совещание по ее обсуждению (и любая необходимая последующая работа) не относится напрямую к разработке продукта. Поэтому никакие задачи, связанные с ней, не должны становиться частью плана итерации.

Будьте конкретны, пусть это станет привычкой

Новые agile-команды нередко не знакомы с написанием автоматизированных модульных тестов или не имеют опыта в этом. Необходимый опыт нарабатывается в процессе первых нескольких итераций. В течение этого периода я предлагаю программистам идентифицировать и оценивать задачи по модульному тестированию в явном виде. Программист может, например, сказать, что кодирование новой функции займет восемь часов и что написание ее модульного теста потребует пять часов. Позднее, когда модульное тестирование войдет в привычку, программист будет заполнять только одну карточку на кодирование новой функции и давать оценку, включающую время на автоматизированное модульное тестирование. Когда что-то вроде модульного тестирования становится привычкой, его можно включать в другую задачу. До этого, однако, оценка в явном виде помогает не забывать о важности данной задачи.

Учет совещаний (которых будет множество)

Необходимо идентифицировать, оценить и включить в задачи совещания, связанные с проектом. При оценке совещания не забудьте учесть время всех участников, а также время, затраченное на его подготовку. Предположим, команда намеревается провести совещание для обсуждения обратной связи от пользователей. На этом часовом совещании собираются присутствовать все семь членов команды, а аналитик планирует потратить два часа на его подготовку. Оценка для этой задачи составляет девять часов. Я обычно включаю такую оценку в план итерации в виде отдельной девятичасовой задачи, а не как отдельные задачи для каждого члена команды.

Программные ошибки

Agile-команды стремятся устранять все ошибки в той итерации, в которой они обнаруживаются. Это становится возможным с приобретением опыта работы в режиме коротких итераций, в частности в результате применения автоматического тестирования. Когда программист дает оценку задачи по кодированию чего-либо, он включает в нее время на устранение ошибок, выявленных в процессе реализации, или выделяет этот процесс и оценивает его как отдельную задачу («Устранение ошибок»). Я предпочитаю представлять это в виде отдельной задачи, но не считаю ее завершенной до тех пор, пока не будут выполнены все тесты.

С дефектом, обнаруженным позже (или не устраненным во время итерации, в которой он был выявлен), поступают точно так же, как и с пользовательской историей. Устранение дефекта следует приоритизировать в последующей итерации так, как приоритизируют любую другую пользовательскую историю. За пределами итерации идея дефекта в целом начинает сходиться на нет. Устранение ошибки и добавление функции становятся двумя разновидностями описания одного и того же.

Подход к работе с взаимозависимостями

Нередко реализация одной пользовательской истории зависит от предварительной реализации другой. В большинстве случаев такие взаимозависимости не представляют значительной проблемы. Обычно существует то, что я считаю естественным порядком реализации пользовательских историй, т.е. имеется последовательность, которая выглядит разумной и для разработчиков, и для владельца продукта.

Проблем не возникает, когда взаимозависимые истории реализуются в естественном порядке. Естественным обычно является тот порядок, который команда принимает при оценке историй. Например, команда SwimStats, скорее всего, будет исходить из того, что добавление пловцов в

систему должно реализовываться раньше, чем их удаление. Если истории реализуются в порядке, отличном от того, что был принят во время оценки, команде во время планирования итерации нередко приходится включать дополнительные задачи, позволяющие работать с историями в новом порядке.

В качестве примера допустим, что естественный порядок для веб-сайта SwimStats предполагает реализацию функций, которые позволяют пользователю добавлять новых пловцов в систему, а затем реализацию функций, позволяющих пользователю просматривать лучшие результаты каждого пловца в соревнованиях. Было бы странно выискивать лучшие результаты до получения окон для добавления пловцов в систему. Это, однако, может быть сделано, если владелец продукта и команда захотят разрабатывать функции в таком порядке. Для этого, конечно, необходимо в достаточной мере проработать базу данных так, чтобы она могла накапливать информацию по пловцам и их результатам. Команде также придется ввести в базу данных информацию как минимум об одном пловце и его результатах. Поскольку это часть функции, которую она не хочет разрабатывать в первую очередь, придется добавлять пловца (и его результаты) в базу данных напрямую, а не через пользовательский интерфейс и разработанную программу.

Чтобы команда SwimStats могла сделать это, во время планирования итерации ей необходимо идентифицировать несколько задач, которые не появились бы, если бы эти две истории реализовывались в естественном порядке. Например, если бы возможность добавлять пловцов уже существовала, команде не нужно было бы включать задачу «Разработка таблиц базы данных для информации об индивидуальных пловцах». Вместе с тем, поскольку истории реализуются не в естественном порядке, такую задачу включить придется.

Означает ли это, что работа не в естественном порядке увеличивает срок выполнения проекта? Ответов может быть два: «не обязательно» и «это не имеет значения».

Во-первых, проект не обязательно потребует больше времени — все, что мы делаем, это перемещаем некоторые задачи из одной пользовательской истории в другую. Разработку таблиц по пловцам в данном примере все равно придется выполнять, раньше или позже. Когда придет время заниматься историей по добавлению новых пловцов в систему, эта история будет реализована быстрее в результате того, что часть работы уже выполнена.

Беспокойство может вызывать влияние перемещения задачи на оценки, данные двум историям. Мы можем, например, переместить работу в один пункт или идеальный день из одной истории в другую. В большинстве случаев это не такое уж значимое событие, и различия сгладятся в ходе

выполнения проекта. Если уж на то пошло, то я вижу здесь пессимистический сдвиг — превращение пятипунктовой истории в шестипунктовую. Однако из-за того, что команда намеревалась завершить только пять пунктов, она немного недооценивает свою скорость. Поскольку влияние небольшого пессимистического сдвига невелико, я обычно не учитываю его. Вместе с тем, если вас беспокоит такое влияние или если сдвиг задачи намного более значителен, переоцените истории, как только вы решите работать с ними в порядке, отличном от естественного.

Во-вторых, даже если работа с историями в таком порядке увеличивает срок выполнения проекта, это не имеет значения, поскольку, надо думать, для отказа от работы в естественном порядке была веская причина. Команда могла изменить порядок работы над историями с тем, чтобы быстрее устранить технический риск. Или владельцу продукта могла раньше потребоваться обратная связь от пользователей по истории, которая при естественном порядке работы была бы реализована позднее. Реализация историй в порядке, отличном от естественного, позволяет команде раньше получить обратную связь и потенциально сэкономить месяц-другой на переделках ближе к концу проекта (когда календарный график сложнее поддается таким изменениям).

Работа, которую трудно разбить

Некоторые функции очень трудно разбить на задачи. Так, недавно мне пришлось заниматься планированием совещания для обсуждения небольшого изменения устаревшей функции. Никто не может быть уверен в своей способности предвидеть все без исключения возможные последствия изменения. Мы знали некоторые фрагменты программы, которые будут затронуты, но у нас не было уверенности в том, что изменение не скажется на каких-либо других фрагментах. Изменения во фрагментах, о которых мы знали, были незначительными, поэтому их оценили в сумме в четыре часа. Если бы затрагивались другие фрагменты, то оценка могла бы быть значительно более высокой — вплоть до 20 часов. Без анализа кода мы ничего не могли сказать, однако прерывать совещание из-за этого не хотелось. В результате мы ввели следующие две задачи:

- Определение того, что затрагивается, — два часа.
- Внесение изменений — 10 часов.

Первую задачу называют спайком. *Спайк* — это задача, которую включают в план итерации специально с целью получения знаний или ответа на вопрос. В нашем случае у команды не было хороших предположений в отношении определенного аспекта, поэтому она создала две задачи: одна — спайк, а другая — заполнитель с предположением о

продолжительности. Спайк помогает команде узнать, как ей подойти к решению другой задачи, позволяющей дать оценку.

Оценка задач

Следующий этап планирования итерации на основе скорости — оценка каждой задачи. Одни команды предпочитают оценивать задачи после того, как все они будут идентифицированы, другие — по мере идентификации. Оценки задач выражаются в идеальном времени. Так, если я считаю, что задача займет у меня шесть рабочих часов, то даю оценку «шесть часов». Я даю такую оценку, даже если шесть часов чистой работы над задачей потребуют от меня затраты целого восьмичасового дня.

Хотя я согласен с общепризнанным мнением, что лучше всего оценивает тот, кто выполняет работу (Lederer and Prasad, 1992), на мой взгляд, оценка в agile-проекте должна быть коллективным занятием. Для этого есть четыре причины.

Во-первых, поскольку задачи не распределяются между конкретными исполнителями во время планирования итерации, на этом этапе нет конкретного исполнителя работы.

Во-вторых, даже если мы предполагаем, кто именно будет выполнять задачу, и даже если этот исполнитель знает больше всех о данной задаче, это не означает, что другим нечего добавить. Допустим, во время совещания по планированию итерации Джеймс говорит: «Мне потребуется примерно два часа, чтобы запрограммировать функцию, — это тривиально!» Однако вы помните, что в прошлом месяце у Джеймса была похожая задача и он дал такой же комментарий, но на выполнение той задачи ушло почти 16 часов. В этот раз, когда Джеймс скажет, что такая задача требует всего лишь два часа, вы можете добавить: «Но, Джеймс, в прошлый раз, когда ты работал над похожей задачей и тоже оценивал ее в два часа, на нее в действительности ушло 16 часов». Скорее всего, Джеймс назовет разумную причину, по которой в этот раз все будет иначе, однако он может согласиться с тем, что с такими задачами связаны определенные трудности, которые он систематически упускает из виду.

В-третьих, высказывание мнения о том, сколько времени может занять та или иная задача, зачастую помогает командам увидеть неправильное представление о пользовательской истории или задаче. Услышав неожиданно высокую оценку, владелец продукта или аналитик может обнаружить, что команда ориентируется на более детальное решение, чем необходимо. Обсуждение оценки членами команды позволяет скорректировать действия до того, как силы и ресурсы будут потрачены впустую.

Наконец, когда оценку дает человек, который будет выполнять работу,

его гордость и самомнение могут помешать ему впоследствии признать ошибочность оценки. При коллективной оценке этот фактор просто исчезает.

Обсуждение дизайна — это нормально

Естественно, создание перечня задач и оценка не обходятся без того или иного обсуждения дизайна. Невозможно сформировать перечень задач, не имея представления о том, как мы собираемся выполнять работу. К счастью, при планировании итерации не обязательно слишком сильно углубляться в дизайн той или иной функции.

Владелец продукта, аналитики и дизайнеры пользовательских интерфейсов могут обсуждать дизайн продукта, необходимую степень реализации функции и то, в каком виде она должна быть представлена пользователям. Разработчики могут обсуждать варианты реализации того, что необходимо. Обе разновидности обсуждения необходимы и уместны. Вместе с тем я никогда не присутствовал на совещании по планированию итерации, где строились бы диаграммы классов или подобные им модели. Желание сделать это служит, пожалуй, лучшим сигналом того, что обсуждение дизайна зашло слишком далеко для этапа планирования итерации. Не загружайте планирование итерации такими обсуждениями.

Совсем не обязательно доходить до проработки дизайна, поскольку все, что требуется на данном этапе, это составление представления о работе, необходимой для реализации конкретных функций. Если вы входите в итерацию и обнаруживаете, что задачи определены неправильно, отбросьте первоначальные задачи и создайте новые. Если ошибочна оценка, зачеркните ее и впишите новое значение. Вписывание задач и оценок в карточки — чудесный подход по той причине, что каждая карточка служит своего рода напоминанием о недолговечности всего.

Подходящий размер для задачи

Создаваемые вами задачи должны иметь примерно такой размер, чтобы каждый разработчик мог завершать в среднем одну из них за день. Подобный размер очень удобен для равномерного выполнения работы в течение всего agile-процесса разработки. Более крупные задачи нередко задействуют одного-двух разработчиков, а остальные члены команды вынуждены ждать, пока те завершат свою работу. Кроме того, если команда проводит короткие ежедневные совещания (Schwaber and Beedle, 2002; Rising, 2002), то однодневные задачи позволяют каждому разработчику отчитываться о завершении как минимум одной задачи практически каждый день.

Естественно, нередко встречаются более крупные задачи. На них, как

правило, следует смотреть как на место для размещения одной или нескольких дополнительных задач, которые добавляются по мере формулирования. Если вам необходимо создать 16-часовую задачу во время планирования итерации, создавайте ее. Вместе с тем, как только появится более глубокое понимание этой задачи, дополняйте или заменяйте ее. Это может означать замену первоначальной карточки карточкой с большим или меньшим временем, чем первоначальные 16 часов.

Планирование итерации на основе обязательств

Подход на основе обязательств является альтернативным способом планирования итерации. Этапы планирования итерации на основе обязательств во многом повторяют этапы планирования итерации на основе скорости. В то же время вместо создания плана с использованием вчерашней погоды для определения количества пунктов или идеальных дней, включаемых в текущую итерацию, команде предлагается добавлять в итерацию по одной истории до тех пор, пока она не исчерпает свои возможности по реализации. В целом подход на основе обязательств представлен на рис. 14.3.

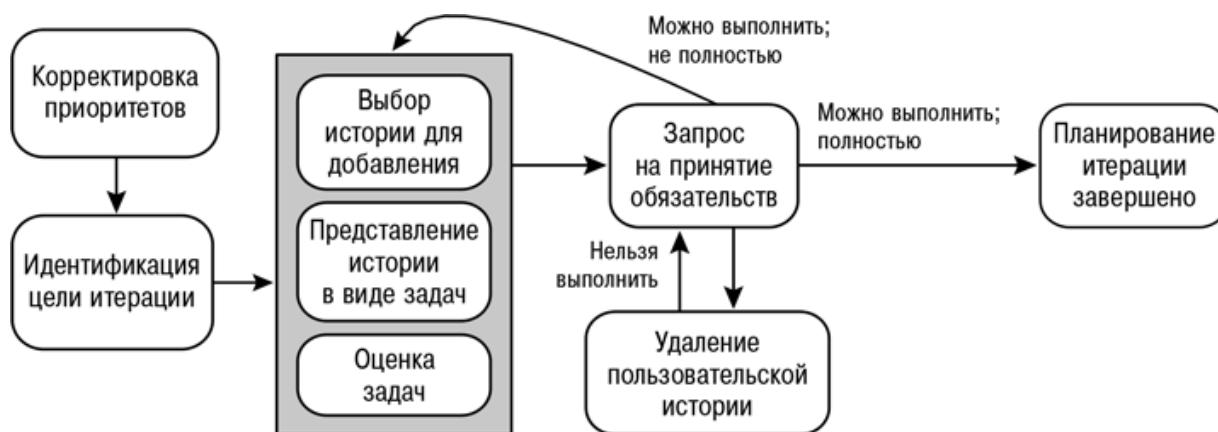


Рис. 14.3. Последовательность действий при планировании итерации на основе обязательств

Первые этапы — корректировка приоритетов и идентификация цели итерации — точно такие же, как и в подходе на основе скорости. Следующий этап, «Выбор истории для добавления», уже отличается. Владелец продукта и команда по-прежнему выбирают историю с наивысшим приоритетом, которая поддерживает цель итерации. Вместе с тем при планировании итерации на основе обязательств истории

выбираются и разбиваются на оцениваемые задачи по одной зараз. Это отличает данный подход от подхода на основе скорости, где выбирается набор историй, оценки которых соответствуют расчетной скорости.

Истории выбираются по одной зараз потому, что после разбивки каждой истории на задачи и оценки этих задач команда решает, сможет ли или не сможет она принять обязательства по реализации этой истории в данной итерации.

Запрос на принятие обязательств

В своем исследовании факторов успеха команд Ларсон и Лафасто (Larson and LaFasto, 1989) установили, что единое обязательство, принятое всеми членами команды, является одним из ключевых факторов успеха. Во время совещания по планированию итерации я спрашиваю команду: «Можете ли вы принять обязательство реализовать функции, которые мы обсудили?» Обратите внимание на то, что я не спрашиваю, смогут ли они принять обязательство реализовать задачи, которые мы идентифицировали. Это совершенно другой вопрос и значительно более слабое обязательство, поскольку оно относится к выполнению набора задач, а не к постановке новой функциональности.

Если в ходе итерации появятся новые задачи (а они почти наверняка появятся), то команда, которая приняла обязательство поставить функциональность, описанную пользовательской историей, будет стараться выполнить также и новые задачи. Команда, которая обязалась выполнить только набор идентифицированных задач, может отказаться от выполнения новых. Не исключено, однако, что новые задачи окажутся слишком объемными для выполнения в данной итерации. В этом случае команде необходимо обсудить ситуацию с владельцем продукта и определить, есть ли возможность реализовать цель итерации. Команде для этого, возможно, придется сократить функциональность истории, а то и полностью исключить ее.

Я спрашиваю команду, сможет ли она принять обязательство по реализации после разбивки каждой пользовательской истории на задачи и оценки этих задач. Применительно к первой пользовательской истории такой вопрос может показаться глупым. В команде, планирующей двухнедельную итерацию, может быть семь человек. Может так случиться, что они идентифицировали работу пока всего лишь на 34 часа, а я спрашиваю, смогут ли они принять обязательство по ее выполнению. Их ответ (либо в словесной форме, либо в виде смущения на лицах), без сомнения, будет следующим: «Конечно, мы можем принять обязательство выполнить это. Нас семь, и у нас целых две недели, а здесь работы всего на 34 часа». Вместе с тем чем дальше продвигается обсуждение и чем больше

пользовательских историй включаются в итерацию, тем более тщательного обдумывания требует мой вопрос о возможности принятия обязательства. В конечном итоге наступает предел, за которым команда уже не может увеличивать объем обязательств. В таком случае она может исключить историю или заменить ее на более мелкую перед завершением обсуждения.

Суммирование оценок

По моему опыту, для определения, сможет ли команда справиться с набором пользовательских историй, лучше всего просуммировать оценки, данные задачам, и посмотреть, представляет ли полученная сумма разумный объем работы. Надо учитывать, конечно, что некоторым задачам присуща высокая неопределенность, поскольку работа не продумывалась детально, а требования туманны. Так или иначе, суммирование оценок дает определенное представление об общем объеме работы.

Допустим, команда работает с использованием двухнедельных итераций. В каждой итерации ей доступно 560 часов ($7 \text{ человек} \times 10 \text{ дней} \times 8 \text{ часов в день}$). Мы знаем, что определенное время тратится на деятельность, не отраженную в карточках с задачами, — ответы на электронные письма, участие в совещаниях и т.д. Также нам известно, что оценки неточны; в конце концов, это *оценки*, а не гарантированные величины. По этим причинам нельзя ожидать, что эта команда возьмется за выполнение задач объемом 560 часов. На практике большинство команд успешно справляются с работой, когда ее плановый объем (сумма оценок на карточках с задачами) составляет от четырех до шести часов в день. Для нашей команды из семи человек, работающей двухнедельными итерациями, это означает плановый объем работы от 280 до 420 часов. В какой точке этого диапазона команда остановится, зависит от того, насколько хорошо она идентифицирует задачи для данной истории, насколько точно оценивает эти задачи, а также от объема сторонних обязательств членов команды и размера общекорпоративных непроизводительных издержек. После пары итераций большинство команд начинают примерно чувствовать, какой объем работы в часах они должны планировать на итерацию.

Прежде чем принимать обязательство выполнить работу во время итерации, необходимо посмотреть на задачи и понять, обеспечивают ли они подходящее распределение работы в соответствии с квалификацией и опытом членов команды. Не получится ли так, что Java-программист окажется перегруженным, а HTML-программисту будет нечего делать? Не окажутся ли выбранные пользовательские истории легко поддающимися программированию, но требующими много времени или трудно тестируемыми, что приведет к перегрузке тестировщика? Не требуют ли выбранные истории предварительного анализа и дизайна пользовательского

интерфейса?

Команда в ситуации вроде этой должна сначала попытаться найти подходы к более рациональному распределению работы. Может ли HTML-программист в этом примере помочь тестировщику? Может ли кто-нибудь другой, кроме дизайнера пользовательского интерфейса, выполнить эту работу? Если нет, то можно ли удалить из этой итерации истории, которые требуют дизайна пользовательского интерфейса, и можно ли вместо них взять истории, в которых это не требуется? Главное, чтобы все в команде отвечали за что-нибудь в пределах своих умений независимо от специализации.

Индивидуальные обязательства

При оценке возможности принять обязательство по реализации набора новых функций некоторые команды предпочитают сначала распределять задачи между конкретными исполнителями, а потом уже оценивать, сможет ли каждый исполнитель справиться с порученной работой. Этот подход работает хорошо, и я, бывало, рекомендовал его использовать (Cohn, 2004). Однако, как оказалось, отказ от распределения задач во время планирования итерации и от индивидуальных оценок, необходимых для принятия обязательств, облегчает команде переход на мышление «мы все работаем над этим вместе».

Если у вас все же возникает необходимость распределить задачи между исполнителями во время планирования итерации, то такое распределение следует считать временным и легко изменяемым в ходе итерации.

Сопровождение других продуктов и обязательство

В дополнение к реализации проекта многие команды также отвечают за поддержку и обслуживание другой системы. Это может быть предыдущая версия продукта, над которым ведется работа, или не связанная с ним система. Когда команда принимает обязательство реализовать набор историй во время итерации, ей необходимо учитывать необходимость обслуживания и поддержки. Я не имею в виду общие задачи по устранению ошибок, которые можно приоритизировать заранее. Они должны проходить процесс приоритизации в ходе планирования итерации. Под обслуживанием и поддержкой я понимаю непредсказуемые, но неизбежные элементы жизни многих команд — поддержку веб-сайта или базы данных, прием обращений за поддержкой от ключевых клиентов, оперативную техническую поддержку и т.д.

Я представляю себе итерацию как пустой стакан. Первое, что наливают в стакан, — это неизменные обязательства команды, такие как поддержка и обслуживание других продуктов. Оставшееся в стакане пространство

доступно для принятия обязательств по выполнению работы во время итерации. Графически этот образ представлен на рис. 14.4. Понятно, что у команды, чей стакан на 10% заполнен работой, связанной с поддержкой, будет больше времени на выполнение другой работы, чем у команды, чей стакан изначально наполнен на 90%.



Рис. 14.4. Объем работ, который команда сможет выполнить во время итерации, зависит от других ее обязательств

В большинстве ситуаций команда не может очень точно предсказать свою будущую нагрузку по поддержке других продуктов. Ей необходимо знать долгосрочное среднее значение, однако 20 часов поддержки в неделю в среднем — это не то же самое, что по 20 часов каждую неделю. Если нагрузка по поддержке и обслуживанию превосходит ожидаемый уровень в течение итерации, у команды может не оказаться возможностей выполнить свои обязательства. Она должна учитывать это и стараться увеличивать обязательства, когда в некоторых итерациях нагрузка по поддержке и обслуживанию оказывается меньше ожидаемой. Такое непостоянство неизбежно у команд, имеющих значительные обязательства по поддержке и обслуживанию.

Мои рекомендации

Эффективны оба подхода — и планирование итерации на основе скорости, и планирование итерации на основе обязательств. Я, однако, предпочитаю подход на основе обязательств. Хотя скорость играет критическую роль в планировании релиза, я не считаю, что она должна играть такую же роль при планировании итерации. Для такого заявления есть две причины.

Во-первых, поскольку скорость является показателем для укрупненных оценок (пункты или идеальные дни), она недостаточно точна для планирования работ в коротких итерациях. Мы можем использовать эти укрупненные оценки для определения общего объема работы, которую команда выполняет во время итерации. Их, однако, нельзя таким же

образом применять для планирования краткосрочной работы в одной итерации.

Во-вторых, команде необходимо реализовывать 20–30 пользовательских историй за итерацию, чтобы ошибки в пунктах или идеальных днях взаимно уравновешивались. Мало какие команды реализуют столько историй за итерацию.

Чтобы показать, к чему приводят эти проблемы, предположим, что команда имела скорость 30 в каждой из последних пяти итераций. Скорость команды практически стабильна, и, скорее всего, в очередной итерации она вновь выполнит 30 пунктов. Вместе с тем мы знаем, что не все пятипунктовые истории одинаковы. Если взять большой набор пятипунктовых историй, мы наверняка сможем найти, скажем, шесть пятипунктовых историй, которые немного легче, чем пять пунктов. Мы можем промахнуться с некоторыми из них, но, если впервые попробуем это, возможно, все получится — скорость может возрасти с 30 до 40. В то же время можно отобрать только те истории, которые немного сложнее, чем пять пунктов. Мы по-прежнему считаем, что их надо оценивать в пять пунктов, однако они немного сложнее, чем остальные пятипунктовые истории.

В проекте мы не перелопачиваем наш набор пользовательских историй в поисках «более легких пятерок» или «более сложных пятерок». Большинство команд, однако, включает от трех до 10 историй в каждую итерацию. При выборе этих нескольких историй для итерации команде наверняка время от времени будет везти или не везти, т.е. будут попадаться, соответственно, более легкие или более сложные истории.

Поскольку за одну итерацию реализуется слишком малое количество историй, чтобы сводить различия на нет, я предпочитаю не использовать скорость при планировании итерации. Вместе с тем из-за того, что эти различия компенсируются в процессе выпуска релиза, скорость очень хорошо работает при планировании релиза.

Соотнесение оценок задач с пунктами

Меня нередко просят объяснить взаимосвязь между оценками задач, используемыми при планировании итерации, и пунктами или идеальными днями, используемыми для более долгосрочного планирования релиза. По моим наблюдениям, команды сбиваются с пути, когда начинают верить в то, что между пунктами и точным числом часов существует сильная взаимосвязь. Так, сравнительно недавно я работал с одной командой, которая отслеживала фактическое количество продуктивных часов на

итерацию и скорость на итерацию. На основе полученных данных она рассчитала, что каждый пункт соответствует примерно 12 часам работы. Команда ошибочно решила, что один пункт всегда равен 12 часам работы. Вместе с тем в реальности мы имеем нечто близкое к тому, что показано на рис. 14.5.

На рис. 14.5 видно, что в среднем реализация однопунктовой пользовательской истории занимает 12 часов. Однако видно также, что одни однопунктовые истории требуют меньше времени, а другие больше. До тех пор, пока команда не оценит лежащие в основе истории задачи, трудно сказать, где конкретная история окажется на кривой, подобной той, что приведена на рисунке.

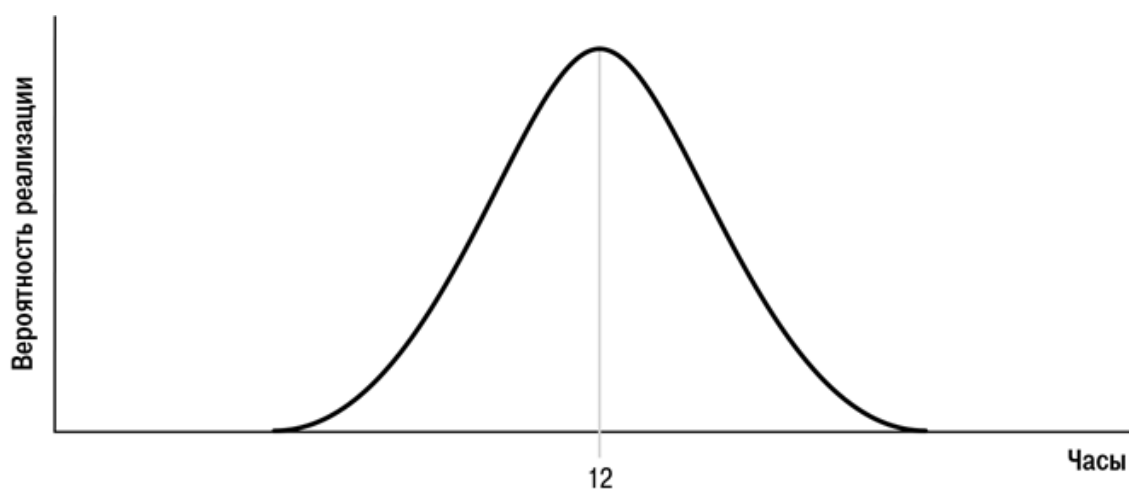


Рис. 14.5. Распределение времени, необходимого для реализации однопунктовой пользовательской истории

Если на рис. 14.5 представлено распределение времени для однопунктовой пользовательской истории, то на рис. 14.6 показано гипотетическое распределение времени для одно-, двух- и трехпунктовой истории. На этом рисунке один пункт эквивалентен в среднем 12 часам. Вместе с тем вполне может оказаться, что реализация некоторых однопунктовых историй занимает больше времени, чем реализация некоторых двухпунктовых историй.



Рис. 14.6. Распределение времени, необходимого для реализации одно-, двух- и трехпунктовой пользовательской истории

В том, что на некоторые двухпунктовые истории требуется меньше времени, чем на некоторые однопунктовые истории, нет ничего необычного, и этого следует ожидать. Это не проблема, поскольку в релизе всегда находятся истории, которые сглаживают такие отклонения, а все участники проекта помнят, что на одни истории уходит больше времени, чем на другие, хотя первоначальные оценки одинаковы.

Дни недели

Когда я только стал руководить agile-проектами, мои команды обычно начинали выполнение итераций в понедельник и завершали их в пятницу. Мы делали это независимо от того, какие итерации использовались — двух-, трех- или четырехнедельные. Понедельник казался самым естественным днем для начала итерации, а пятница — для ее завершения. Мои представления, однако, изменились после того, как я взялся за обучение команды, разрабатывавшей веб-сайт, который активно использовался в будни и практически не посещался в выходные.

У этой команды наиболее логичным моментом для обновления сайта был вечер пятницы. Если что-то шло не так, она могла устранить обнаруженную ошибку за выходные и последствия этой ошибки были минимальными, поскольку сайт в это время посещался очень редко. Чтобы приспособиться к такому режиму, команда решила использовать двухнедельные итерации, которые начинались в пятницу и завершались в четверг. Этот режим оказался просто идеальным. Пятница посвящалась анализу завершенной итерации и планированию следующей. Чаще всего на это уходила первая половина дня, а потом команда бралась за выполнение новой итерации или иногда устраивала поход в бар или в боулинг-клуб. Полномасштабная работа над итерацией начиналась в следующий понедельник. Это было очень удобно потому, что в понедельник никто уже не отвлекался на совещания и т.п.

Иногда команда утром в пятницу подчищала хвосты — завершала появившуюся в последний момент работу, которую не успела сделать в четверг. Это не было правилом и случалось от силы пару раз, но возможность завершить такую работу за несколько часов в пятницу утром намного лучше перспективы доделывать ее в выходные (что было бы неизбежным, если бы итерации начинались в понедельник).

Резюме

В плане итерации, в отличие от плана релиза, более детально прорабатывается конкретная работа, выполняемая в рамках отдельной итерации. Если горизонт типичного плана релиза составляет от трех до девяти месяцев, то горизонт планирования итерации не выходит за пределы отдельно взятой итерации. Относительно большие пользовательские истории из плана релиза разбивают в плане итерации на задачи. Каждую задачу оценивают в количестве идеальных часов, которые требуются для ее выполнения.

Существует два основных подхода к планированию итерации: планирование на основе скорости и планирование на основе обязательств. У этих подходов много общих этапов, а их применение нередко приводит к одному и тому же результату, т.е. дает один и тот же план итерации.

Вопросы для обсуждения

1. Какой подход к планированию итерации вы предпочитаете: на основе скорости или на основе обязательств? Почему?
2. Как вы относитесь к идее отказа от распределения конкретных задач между членами команды во время планирования итерации?

Глава 15

Выбор длины итерации

*Все неопределенно до такой степени, что ты не осознаешь
этого до тех пор, пока не попытаешься дать точное
определение чему-либо.*

Бертран Расселл

Подавляющее большинство agile-процессов и команд, использующих их, ориентированы на итерации длиной от двух до четырех недель. Есть команды, которые используют и более длинные итерации, но две–четыре недели — это приемлемый стандарт для большинства команд. Какой-то одной волшебной продолжительности итерации, которая подходит для всех команд при любых обстоятельствах, не существует. У одной и той же команды длина итерации, подходящая для одного проекта, не обязательно подходит для другого проекта.

Факторы, влияющие на выбор длины итерации

При выборе длины итерации вы должны руководствоваться следующими факторами:

- Длина релиза, над которым ведется работа.
- Уровень неопределенности.
- Простота получения обратной связи.
- Продолжительность сохранения неизменности приоритетов.
- Готовность продолжать работу без получения внешней обратной связи.
- Издержки итерационного подхода.
- Быстрота появления ощущения цейтнота.

Какой-либо предопределенной относительной важности этих факторов не существует. Важность каждого из них полностью зависит от контекста проекта.

Общая длина релиза

Короткие проекты выигрывают от использования коротких итераций. От длины итераций проекта зависят следующие аспекты:

- Частота демонстрации программного продукта (в потенциально готовом виде) пользователям и клиентам. Конечно, программу можно показать публике и в середине итерации, но она обычно приобретает потенциально готовую форму только к концу итерации.
- Частота измерения прогресса. Представление о скорости прогресса команды можно получить и в процессе итерации, однако только к ее концу можно достоверно измерить объем выполненной работы.
- Частота уточнения направления работы владельцем продукта и командой, поскольку приоритеты и планы корректируются между итерациями.

Если команда работает над релизом, который необходимо представить через три месяца, то одномесечные итерации позволят лишь два раза получить обратную связь, определить прогресс и скорректировать направление работ. В большинстве случаев этого недостаточно.

Мой опыт говорит о том, что в течение проекта такая возможность должна появляться не менее четырех-пяти раз. Таким образом, если общая продолжительность проекта составляет четыре месяца и более, то вполне можно рассмотреть целесообразность использования месячных или четырехнедельных итераций. Если же общая длина релиза меньше, то в проекте необходимо использовать пропорционально более короткие итерации.

Уровень неопределенности

Неопределенность имеет множество проявлений. Нередко неопределенностью характеризуются конкретные потребности клиента или пользователей, будущая скорость команды, а также технические аспекты проекта. Чем больше неопределенность любого типа, тем короче должны быть итерации. Когда значительная неопределенность связана с выполняемой работой или с создаваемым продуктом, короткие итерации позволяют команде чаще измерять прогресс через определение скорости и чаще получать обратную связь от других участников проекта, клиентов и пользователей.

Простота получения обратной связи

Длину итерации следует выбирать так, чтобы максимально повысить объем,

частоту и своевременность получения обратной связи всей командой. В зависимости от окружающей среды итерации могут быть длиннее или короче. В одних организациях очень легко получить неофициальную обратную связь от внутренних заинтересованных лиц или пользователей на протяжении всей итерации, но крайне трудно привлечь этих лиц к участию в плановом совещании по анализу итерации после ее завершения. В других организациях проблема прямо противоположна — трудно получить обратную связь на ежедневной основе, однако заинтересованные лица, пользователи и другие участники проекта с готовностью приходят на плановое официальное совещание по анализу итерации (особенно если там предлагают какое-нибудь угощение).

Выбирайте длину итерации так, чтобы максимизировать ценность обратной связи, которую можно получить от инсайдеров и аутсайдеров организации.

Продолжительность сохранения неизменности приоритетов

После того как команда разработчиков принимает обязательства по реализации определенного набора функций за итерацию, очень важно, чтобы она не отклонялась от этой цели. Таким образом, важно, чтобы владелец продукта не изменял приоритеты во время итерации и чтобы он помогал защищать команду от других, кто может попытаться изменить приоритеты. Как результат, продолжительность сохранения неизменности приоритетов становится существенным фактором при выборе длины итерации.

Ключевым соображением является время, необходимое для превращения хорошей новой идеи в работающую программу. Возьмем для примера команду, использующую четырехнедельные итерации. Если предположить, что новые идеи с одинаковой вероятностью могут возникнуть в любой момент в процессе осуществления итерации, то в среднем появление новой идеи должно приходиться на середину итерации. Эта новая идея будет приоритизирована в следующей итерации, которая начнется через две недели. Еще четыре недели (полная итерация) потребуется на то, чтобы новая идея превратилась в потенциально готовую работающую программу. Этот процесс представлен на рис. 15.1. Ключевой момент, который необходимо вынести из этого примера, заключается в том, что время с момента появления новой идеи до создания работающей программы составляет в среднем полторы длины итерации, используемой командой.

Как говорилось в главе 14 «Планирование итерации», команда, на которой лежат обязанности по обслуживанию или поддержке, помимо

новой разработки, всегда резервирует определенное время на работы, связанные с обслуживанием и поддержкой. На рис. 15.1 показана реальная ситуация, в которой некто предлагает команде идею, не укладывающуюся в ее резерв на обслуживание и поддержку.

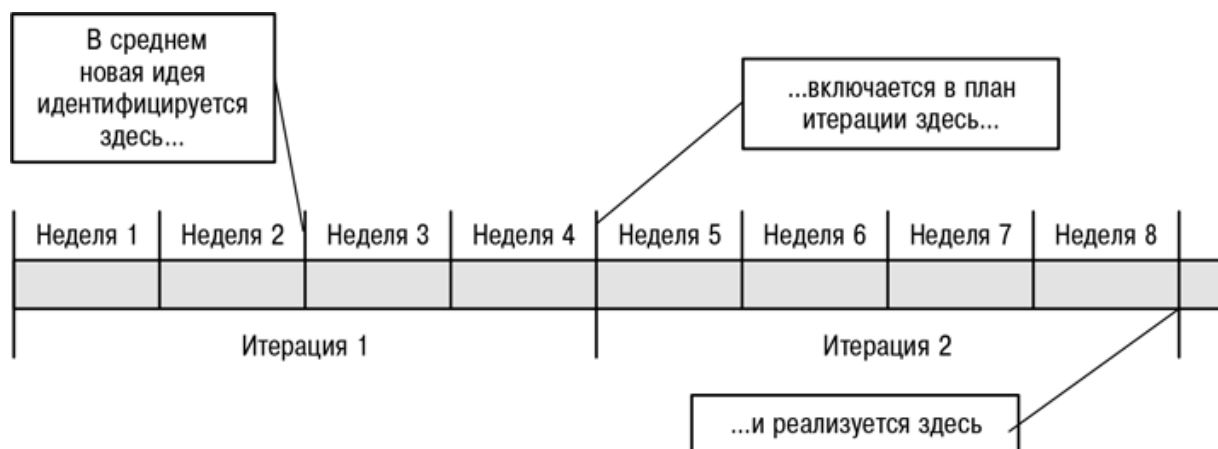


Рис. 15.1. Для превращения идеи в программу в среднем необходимо полторы итерации

Готовность продолжать работу без получения внешней обратной связи

Даже у хорошо мотивированной и открытой для взаимодействия команды результаты итерации могут оказаться никому не нужными при их демонстрации внутри организации или внешним пользователям. Такое может произойти, если разработчики неправильно поняли владельца продукта (и недостаточно часто общались с ним в ходе итерации). Это также случается, если владелец продукта неправильно понял потребности рынка или пользователей. Потеря практически никогда не считается компенсированной, если мы не научились на ней чему-либо. Вместе с тем чем реже команда получает внешнюю обратную связь, тем выше вероятность того, что она собьется с пути, и тем больше размер потерь, если они произойдут.

Издержки итерационного подхода

С каждой итерацией связаны определенные издержки. Например, каждая итерация должна полностью тестироваться после завершения. Если это очень затратно (обычно с точки зрения времени), команда может предпочесть более длинные, четырехнедельные итерации. Естественно, одной из целей успешной agile-команды является сокращение (или почти полное устранение) издержек, связанных с каждой итерацией. Эти издержки могут быть значительными, особенно в первых итерациях, и,

таким образом, влиять на выбор длины наиболее подходящей длины итерации.

Быстрота появления ощущения цейтнота

Мой коллега Нильс Малото (Malotauх, 2004) подчеркивает, что «пока дата окончания проекта где-то далеко в будущем, на нас ничто не давит и мы работаем неторопливо. Когда же дата завершения появляется на горизонте, мы начинаем работать интенсивнее. Дата завершения даже четырехнедельных итераций не так уж далека. Однако до нее достаточно много времени, чтобы многие команды ощущали заметно меньшее напряжение в течение первой недели, чем в течение четвертой и последней недели итерации.

Решением в этом случае является выбор такой длины итерации, которая выравнивает давление, ощущаемое командой. Идея не в том, чтобы усилить давление на команду («Вы *должны* закончить работу сегодня!»). Речь идет о более равномерном распределении нормального стресса по итерации подходящей длины.

Не меняйте выбранной длины, если хотите добиться стабильного ритма

Какую бы длину итерации вы ни выбрали, лучше придерживаться ее и не менять то и дело. При использовании постоянной длины итерации у команд вырабатывается естественный ритм. Когда я начал заниматься agile-разработкой и применять один из ранних вариантов методологии Scrum (Takeuchi and Nonaka, 1986; DeGrace and Stahl, 1990), мои команды обычно выбирали длину каждой итерации в зависимости от объема работ, который предполагалось выполнить. За двухнедельной итерацией могла следовать шестинедельная итерация, а за ней — четырехнедельная и т.д. Опыт, полученный в результате выполнения множества проектов, показал, что команды намного лучше подгоняют объем работ к длине итерации, а не длину итерации к объему работ.

Стабильный ритм выполнения итерации — это своего рода сердцебиение проекта. Коллега Саймон Бейкер, agile-тренер в think-box ltd., говорит об этом так: «Как сердцебиение, стабильность которого необходима для жизнедеятельности организма, фиксированная длина итерации является константой, помогающей задать ритм разработки (и поставки). Ритм в моей практике — значимый фактор, который позволяет поддерживать устойчивый темп» (Baker, 2004).

Принятие решения

Одна из главных целей при выборе длины итерации заключается в том, чтобы найти такую длину, которая будет способствовать работе каждого в постоянном темпе на протяжении всей итерации. Если длина будет слишком большой, возникает естественный соблазн немного расслабиться в начале итерации, а это ведет к панике и сверхурочной работе в ее конце. Стремитесь подобрать такую длину итерации, которая сглаживает подобные крайности.

Я экспериментировал с разными длинами итерации и остановился на двух неделях. Однонедельные итерации (или еще более короткие) могут быть очень сумбурными и напряженными. Срок сдачи работы отстоит от начала итерации всего на несколько дней. Слишком короткие итерации не оставляют времени на спасение ситуации в случае, если кто-то вдруг заболит или что-то пойдет не так. При отсутствии в проекте уже обкатанных полностью автоматизированных процессов тестирования всех частей системы я почти никогда не рекомендую начинать с однонедельных итераций.

Четырехнедельная итерация, в свою очередь, вызывает чувство уймы времени у тех, кому приходилось работать с использованием одно- и двухнедельных итераций. Мой опыт показывает, что при четырехнедельных итерациях, в отличие от более коротких итераций, у команды нередко появляется время для поиска и реализации более креативных решений. Опытная agile-команда на стадии проекта, которая предполагает значительный объем экспериментирования и исследований, может выиграть от использования четырехнедельной итерации. Вместе с тем в итерации с такой продолжительностью возникает очень четкое ощущение начала, середины и конца. Мне лично не нравятся переходы от неторопливого начала к лихорадке завершающего этапа.

На мой взгляд, идеальными являются двухнедельные итерации. Издержки, связанные с планированием и тестированием, значительно лучше поддаются управлению, когда они распределяются по двухнедельному периоду. Первая неделя двухнедельной итерации может восприниматься иначе, чем вторая неделя, но эта разница не настолько значительна, как в случае четырехнедельной итерации. Кроме того, большинство организаций могут (при достаточном опыте) научиться не менять приоритеты в течение двух недель, тогда как удержаться от этого на протяжении четырех недель очень сложно.

$$6 \times 2 + 1$$

Непрерывная работа с использованием двухнедельных итераций может

сильно напрягать команду в результате постоянной необходимости выдавать готовый продукт в условиях, когда срок сдачи работы не выходит за пределы следующей недели. Мой любимый прием снижения этого напряжения — работа макроциклами из шести двухнедельных итераций, за которыми следует однонедельная итерация. Я обозначаю этот цикл как « $6 \times 2 + 1$ ». Во время двухнедельных итераций команда работает над элементами, приоритизированными владельцем продукта. Для однонедельной итерации команда подбирает работу самостоятельно. Это не означает, что она использует это время для игр или проводит неделю на пляже. Члены команды фокусируются в течение этой недели на той работе, которую они считают приоритетной для проекта. Программисты могут заняться реорганизацией кода, выполнять которую в середине другой итерации, на их взгляд, рискованно, или поэкспериментировать с новыми технологиями. Тестировщик может заняться автоматизацией существующих ручных тестов. Аналитик может заняться исследованием следующей крупной функции, которой, по его мнению, было уделено недостаточно внимания.

Возможность получить неделю для работы над тем, что команда коллективно приоритизировала для себя, может служить очень сильным воодушевляющим фактором. Важно подчеркнуть, что такая однонедельная итерация ни в коей мере не свалка для недоделок с предыдущих итераций. Это время, когда команда может поработать над тем или иным техническим долгом, который возник в первых итерациях в процессе освоения agile-подхода или в еще более ранние времена.

Два примера

Чтобы понять, как применять эти факторы, рассмотрим две команды: команду проекта Нара и команду проекта Goodman. Эти команды совершенно реальны, изменены лишь некоторые детали.

Проект Нара

Команда проекта Нара в составе семи человек разрабатывала клиент-серверное приложение для настольных компьютеров. Приложением должны были пользоваться 600 работников компании. Приложение не предполагалось продавать или использовать за пределами компании. Пользователи располагались в трех городах, в одном из которых находилась команда разработчиков в полном составе. Идея продукта зародилась в процессе обновления существующей системы, обслуживание которой стало слишком дорогим. Как следствие изменений в ключевом бизнесе компании,

в проект запланировали включить большое количество новых функций. По оценкам, проект должен был занять 13 месяцев, однако быстрый рост компании сделал необходимым ранний выпуск релиза, пусть даже с ограниченной функциональностью.

Для проекта Нара команда выбрала четырехнедельные итерации. Мы знали, что выполнение проекта займет не менее шести месяцев, поэтому даже четырехнедельные итерации давали массу возможностей привести программу в состояние потенциальной готовности к выпуску релиза, который можно передать реальным пользователям. Проект характеризовался довольно большим, но не чрезмерным набором требований и неопределенностью технологии. Все разработчики имели большой опыт в использовании текущих технологий (C++ и Oracle). Хотя новое приложение должно было иметь функции, значительно выходящие за пределы возможностей существующего, старое приложение приняли за базовую модель того, что необходимо.

Проектная команда установила связь со многими целевыми пользователями системы. Большинство пользователей были готовы принять участие в обсуждении того, как должна выглядеть новая система. Вместе с тем компания развивалась так быстро, что доступ к этим пользователям оказался в определенной мере ограниченным. Мы не могли отнимать у них слишком много времени. Четырехнедельные итерации очень хорошо подходили для такой ситуации. В этих условиях представление пользователям новых версий каждые две недели было слишком частым. Выпуск новой версии раз в четыре недели позволял нам привлечь больше пользователей к экспериментированию с программой в изолированной среде, созданной специально для этой цели.

Поскольку приложение было совершенно новым, издержки итерационного подхода оказались незначительными и не влияли на принятие решения. Проект был критически важным для закрепления успеха компании, поэтому находился под постоянным наблюдением со стороны руководства, начиная с генерального директора. По большей части нам удавалось определять и сохранять приоритеты на протяжении четырех недель. Даже при такой длине итерации нас не покидало ощущение цейтнота по той причине, что команда ясно осознавала необходимость как можно скорее предоставить пользователям начальный релиз.

Проект Goodman

Команда проекта Goodman работала над первой версией коммерческого приложения для корпораций. В течение первых трех лет компания предполагала продать не более 5000 лицензий на программу. Вместе с тем продукт был дорогим, со средней ценой \$50 000 на пользователя.

Разработчиков, общая численность которых составляла 18 человек, разбили на две скоординированно работающие команды. Ожидалось, что выполнение проекта Goodman займет один год, однако предварительный релиз планировали передать небольшому числу клиентов через шесть месяцев.

Для проекта Goodman команда выбрала двухнедельные итерации. Поскольку мы задались целью выпустить первоначальный релиз через шесть месяцев, команда вполне могла бы использовать четырехнедельные итерации, однако этот проект характеризовался крайне высокой неопределенностью. Компания считала, что знает круг потенциальных пользователей продукта, однако вокруг этого представления время от времени вспыхивали споры. Было непонятно, что именно должен представлять собой продукт — профессиональную дорогую систему, как предполагалось изначально, или более дешевую систему, предназначенную для широкой аудитории. Решение вроде бы было принято, но его изменили, а потом изменили еще раз, однако команда так и не обрела уверенность в том, что она получила ответ. Кроме того, очень значительной была доля спонтанных требований типа «Я скажу, как должно быть, когда увижу это».

По этому сложному проекту было очень затруднительно получить обратную связь — в силу коммерческого характера продукта у нас отсутствовали внутренние пользователи. Многие крупнейшие, а также потенциальные клиенты компании находились за границей, что усложняло ситуацию. В компании все же были люди, работающие с целевыми клиентами и способные стать вероятными пользователями продукта, поэтому мы приняли их за прообраз реальных пользователей. Из-за неопределенности и изменчивости этого проекта нам требовалась предельно частая обратная связь и, как следствие, более короткие итерации.

В таких условиях приоритеты не могли оставаться неизменными более нескольких дней, а это также требовало сокращения длины итераций. Именно поэтому мы остановились на двухнедельных итерациях. В этом проекте с самого начала существовала команда по автоматизации тестирования, которая действовала очень эффективно. Это помогло снизить итерационные затраты и успешно работать с короткими итерациями. Наконец, в результате того, что компания совсем недавно превратилась из стартапа в публичную компанию, нам было очень важно поддерживать ощущение цейтнота. Компания осуществила публичное размещение акций, пообещав скорый выпуск успешной программы. Короткие двухнедельные итерации помогали нам сконцентрироваться на предельно быстрой разработке этой программы.

Избегайте конца квартала

Несмотря на соблазн совместить итерации с концом месяца, я всеми силами стараюсь не допустить этого. Если вы привяжете итерации к концу месяцев, одна из каждых трех итераций будет совпадать с концом финансового квартала. Хотя это не такая уж проблема в частных компаниях, этого нельзя сказать о публичных компаниях, для которых очень важно выполнение квартальных целевых показателей по выручке.

Мне довелось участвовать в одном проекте, в котором выпуск нового релиза был запланирован на пятницу 31 марта 2000 г. Этот релиз был ключевой целью девятимесячного периода работы компании (такой срок является предельным для отдельно взятого релиза). За две недели до срока выпуска релиза наш владелец продукта отправился в отпуск со своими детьми школьного возраста. Находясь в Диснейленде, он доблестно пытался решить по телефону несколько очень важных для нас вопросов. Однако его отсутствие пришлось на критический момент, и нам так и не удалось завершить определенную работу в последней итерации перед выпуском крупного релиза.

После возвращения владельца продукта мы смогли решить все оставшиеся вопросы в течение более короткой однонедельной итерации. Это отодвинуло дату выпуска релиза с 31 марта на 7 апреля. Хотя задержка на одну неделю не кажется критической для проекта со сроком выполнения девять месяцев, на практике она привела к переносу поставки продукта с одного квартала на другой, а это имело огромное значение для публичной компании. Из-за того, что плановая поставка нескольких сотен копий, намеченная на 31 марта, не состоялась, выручку от продаж и обновлений уже нельзя было признать в первом квартале. С таким же успехом эту поставку можно было осуществить 30 июня, а не 7 апреля.

С тех пор я больше не планирую выпуск релизов на конец месяца. Разработка программного обеспечения связана с таким количеством неизвестных и неопределенностей, что мне не хочется терять возможность своевременного признания выручки из-за того, что выпуск релиза планируется на конец квартала.

Резюме

Большинство agile-команд используют итерации длиной от двух до четырех недель. Универсальной длины итерации, подходящей всем командам, не существует. Каждая команда должна исходить из конкретной ситуации при выборе подходящей длины итерации. В число факторов, влияющих на это решение, входят:

- длина релиза, над которым ведется работа;
- уровень неопределенности;

- простота получения обратной связи;
- продолжительность сохранения неизменности приоритетов;
- готовность продолжать работу без получения внешней обратной связи;
- издержки итерационного подхода;
- быстрота появления ощущения цейтнота.

Вопросы для обсуждения

1. Какая длина итерации больше всего подходит для вашего текущего проекта?
2. Что изменится в вашем проекте при использовании однонедельных итераций? Что изменится в случае использования двухмесячных итераций?

Глава 16

Оценка скорости

Лучше быть примерно правым, чем точно неправым.

Джон Мейнард Кейнс

Одной из проблем при планировании релиза является оценка скорости команды. Существует три подхода к ее решению:

- использовать исторические значения;
- выполнить одну итерацию;
- воспользоваться прогнозом.

Бывает, что применимы все три подхода. Так или иначе, несмотря на выбранный подход, при оценке скорости результат следует представлять в виде диапазона. Допустим, вы оценили скорость команды в определенном проекте как 20 идеальных дней на итерацию. Шансы на то, что вы дали правильную оценку, очень ограничены. Скорость может составлять 21, 19 и даже 20,0001. Поэтому не говорите, что скорость равна 20, а давайте диапазон, например скажите, что ваша скорость находится в интервале между 15 и 24.

В последующих разделах я опишу каждый из трех общих подходов — использование исторических величин, выполнение итерации и использование прогноза — и дам свои рекомендации по выбору подходящего диапазона.

Использование исторических значений

Здорово, когда есть исторические значения. Проблема исторических значений в том, что их ценность наиболее высока, когда новый проект и команда мало чем отличаются от старого проекта и команды в прошлом. Любые изменения в составе команды или в технологии снижают полезность исторических показателей скорости. Прежде чем использовать их, задайте себе примерно такие вопросы:

- Осталась ли прежней технология?
- Осталась ли прежней область деятельности?
- Осталась ли прежней команда?
- Остался ли прежним владелец продукта?
- Остались ли прежними инструменты?
- Осталась ли прежней рабочая среда?
- Были ли эти оценки сделаны теми же людьми?

Ответы на эти вопросы нередко утвердительны, если команда переходит к работе над новым релизом продукта, над которым она только что работала. В этом случае использование исторических показателей команды совершенно уместно. Так или иначе, даже несмотря на то, что в такой ситуации скорость относительно стабильна, ее все равно необходимо представлять в виде диапазона. Создать его можно простым прибавлением и вычитанием нескольких пунктов из среднего значения. Можно также посмотреть на лучшие и худшие итерации за последние два или три месяца.

Если же ответ на любой из перечисленных выше вопросов отрицателен, то следует дважды подумать, прежде чем использовать исторические скорости. Как вариант, на них можно опереться, но при этом увеличить диапазон для отражения неопределенности, присущей такой оценке. С этой целью начните с расчета средней скорости команды в течение работы над предыдущим релизом. Если команда выполнила работу объемом 150 пунктов за 10 итераций, то ее средняя (среднеарифметическая) скорость равна 15 пунктам.

Прежде чем читать о том, как преобразовать этот показатель в диапазон, посмотрите на рис. 16.1. На нем показан конус неопределенности, о котором мы говорили в главе 1 «Цель планирования». Из конуса неопределенности следует, что фактический срок проекта лежит в диапазоне между 60% и 160% от того, чему он равен в соответствии с нашими представлениями. Поэтому, чтобы преобразовать нашу среднюю скорость в диапазон, я умножаю ее на 60% и на 160%[\[6\]](#). Таким образом, если наша историческая скорость равна 15, то я говорю, что скорость лежит в диапазоне от 9 до 24.

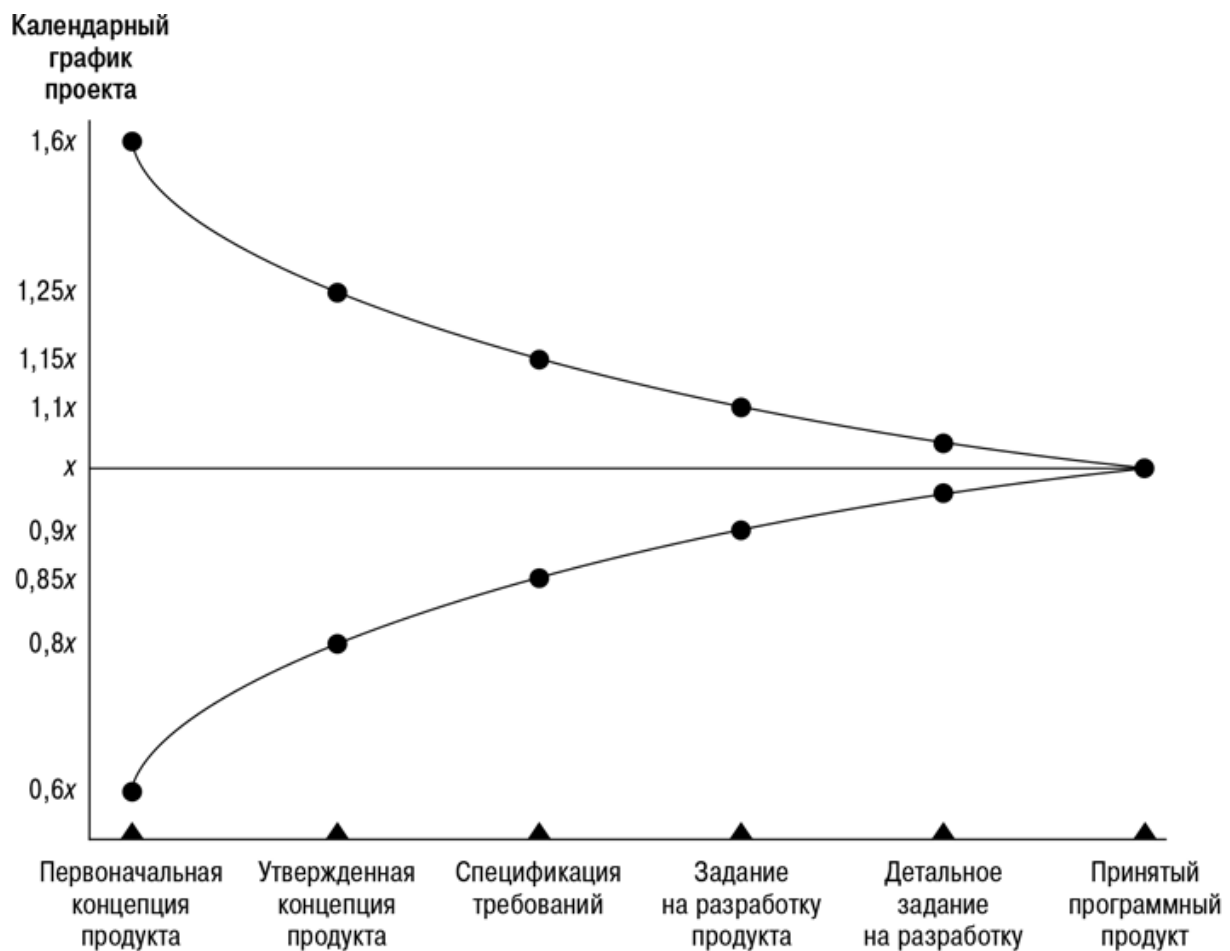


Рис. 16.1. Конус неопределенности вокруг оценок в календарном графике

Этот диапазон может показаться большим, однако с учетом неопределенности в данной точке он приемлем. Создание диапазона подобным образом помогает проектной команде следовать совету, данному в эпиграфе к этой главе, т.е. быть примерно правой и не оказаться точно неправой. Широкий диапазон значений ожидаемой скорости облегчает попадание в него.

Выполнение итерации

Идеальным способом спрогнозировать скорость является выполнение итерации (или двух-трех итераций) с последующей оценкой значения на основе *наблюдаемой скорости*. Поскольку лучшим способом предсказания скорости является ее наблюдение, этот подход следует выбирать по умолчанию. Во многих традиционных проектах разработчики создают «очевидные» требования или инфраструктуру, аналитики «уточняют» требования, а руководитель проекта составляет полный перечень задач, который становится планом проекта. В agile-проекте все это требует

времени, нередко нескольких итераций.

Не так давно я консультировал директора по разработке, который сказал, что в его компании окончательные сроки для традиционных проектов продолжительностью один год не устанавливаются до истечения двух месяцев. Именно столько требуется им для того, чтобы «зафиксировать» требования и сформировать план. Он рассказал, что даже после таких усилий как минимум 50% пунктов их проектных планов, а зачастую и больше, не отличаются точностью. Мы договорились, что он выделит этот период команде для свободной работы над проектом, будет наблюдать за скоростью команды в течение двух-трех итераций, а затем использует эту скорость для определения даты релиза.

По тем же причинам, как и в случае директора по разработке, большинству руководителей проектов полезно воздерживаться от оценок на протяжении как минимум одной итерации. Если вы оказались в такой ситуации, воспользуйтесь возможностью выполнить итерацию и измерить скорость. Затем определите диапазон относительно ее значения с помощью конуса неопределенности. Так, если вы выполните итерацию и получите скорость 15, преобразуйте ее путем умножения на 0,6 и 1,6 в диапазон 9–24.

Если команда может выполнить три итерации или более, прежде чем оценивать скорость, то у нее появляется пара дополнительных возможностей определения диапазона. Во-первых, она может просто использовать диапазон наблюдаемых значений. Допустим, команда выполнила три итерации и получила скорости 12, 15 и 16. В этом случае скорость будет лежать в диапазоне от 12 до 16.

Во-вторых, она может применить конус неопределенности. Хотя у подхода, который я собираюсь описать, нет солидной эмпирической основы, он работает и дает результаты. Вот этот подход: рассчитайте среднюю скорость для выполненных итераций. Затем для каждой итерации сместитесь на один этап вправо на конусе неопределенности. Если команда выполнила одну итерацию, используйте диапазон для этапа «начальная концепция продукта». Если команда выполнила две итерации, используйте диапазон для этапа «утвержденная концепция продукта» (от 80% до 120%) и т.д. Для удобства значения нужных множителей приведены в табл. 16.1.

**Таблица 16.1. Множители для оценки скорости
в зависимости от количества выполненных итераций**

Количество выполненных итераций	Нижний множитель	Верхний множитель
1	0,60	1,60
2	0,80	1,25
3	0,85	1,15
4 и более	0,90	1,10

Предположим, что команда выполнила три итерации при средней скорости 20 за период. Для трех итераций диапазон составляет 85–115%. Это означает, что фактическая скорость к концу проекта будет, скорее всего, лежать в диапазоне от 17 до 23.

Обычно я не распространяю этот анализ далее трех или четырех итераций. Я не использую конус неопределенности, например, для создания видимости точного знания скорости после шести итераций и получения уверенности в том, что она не изменится до конца проекта.

Некоторые организации не хотят начинать проект без получения более или менее конкретного представления о том, сколько времени он займет. В таких случаях подчеркивайте, что необходимость предварительного выполнения нескольких итераций связана не с желанием избежать оценки вообще, а со стремлением избежать оценки без адекватной основы. Подчеркивайте, что целью этих первых итераций является оценка темных уголков системы, более глубокое понимание используемых технологий, получение представления о требованиях и измерение быстроты прогресса, обеспечиваемого командой.

Прогнозирование скорости

Бывает, что исторические данные отсутствуют, а посвятить несколько итераций наблюдению за скоростью нет возможности. Допустим, оценка нужна для проекта, который начнется не раньше чем через 12 месяцев. Или, как вариант, проект может начаться скоро, но только после подписания клиентом договора на выполнение работы. Такие случаи характеризуются двумя ключевыми особенностями. Во-первых, с учетом желания минимизировать затраты на проект вы вряд ли будете проводить итерации по проекту, который начнется в отдаленном будущем или может не начаться вообще. Во-вторых, оценки скорости по таким проектам должны отражать высокую степень неопределенности.

В подобных ситуациях приходится прогнозировать скорость.

Прогнозирование скорости редко имеет преимущество перед другими методами, однако это важная возможность, которую необходимо всегда иметь в запасе. Лучше всего прогнозировать скорость на основе развертывания пользовательских историй до составляющих задач, оценки этих задач (как это делается в процессе планирования итерации), определения подходящего для итерации объема работ и расчета скорости, которой можно было бы достичь в случае выполнения этих работ за итерацию. Процесс прогнозирования имеет следующие этапы:

1. Оценка количества часов, в течение которых каждый участник сможет ежедневно заниматься проектом.
2. Определение суммарного количества часов, которые будут затрачены на проект в процессе итерации.
3. Произвольный и в определенной мере случайный выбор историй и их разбивка на составляющие задачи. Повторяется до тех пор, пока не будет идентифицировано достаточно задач для заполнения доступных на итерацию часов.
4. Преобразование найденной на предыдущем этапе скорости в диапазон.

Посмотрим на примере, как работает такой подход.

Оценка количества доступных часов

Практически каждый из нас имеет какие-либо дополнительные обязанности, помимо своих прямых обязанностей по конкретному проекту. Все мы отвечаем на электронные письма и телефонные звонки, участвуем в корпоративных совещаниях и т.п. Количество времени, посвящаемое дополнительным обязанностям, варьирует от человека к человеку и от организации к организации. Так или иначе, участники проекта обычно не могут посвящать 100% своего времени работе над проектом.

По моим наблюдениям и по мнению моих коллег, большинство работников, полностью занятых в проекте, могут уделять проекту от четырех до шести часов в день. Это соответствует данным о том, что участники уделяют работе над проектом от 55% (Ganssle, 2004) до 70% (Boehm, 1981) своего времени. Самый высокий показатель, по данным Кеннеди (Kennedy, 2003), у инженеров компании Toyota, где высокоэффективный процесс бережливого производства позволяет посвящать работе над целевым проектом до 80% времени.

Воспользуемся этими данными в целях оценки количества времени, которое члены проектной команды могут ежедневно посвящать работе над проектом. Если вы работаете в крупной бюрократической организации, то

ваш показатель будет, скорее всего, низким. Если же вы сотрудник гаражного стартапа из трех человек, то ваш показатель, скорее всего, будет высоким. В целях нашего примера предположим, что, по оценке команды SwimStats, они смогут уделять проекту по шесть часов в день.

Оценка времени, доступного в итерации

Этот этап несложен: нужно просто умножить количество доступных часов в день на количество членов команды и на количество дней в итерации. Допустим, в состав команды SwimStats входит один аналитик, один программист, один администратор баз данных и один тестировщик. Четыре человека, каждый из которых работает по шесть часов в день, вместе будут отрабатывать 24 часа в день. За десятидневную итерацию они посвятят проекту примерно 240 часов.

Когда я внедрял такой подход, некоторые команды хотели учесть, помимо прочего, время отпусков, время отсутствия работников по болезни и другие перерывы в работе. Не стоит заморачиваться — это вряд ли сделает расчет более точным. Такие события — одна из причин, по которым никто не строит планы с расчетом на 100%-ную доступность команды.

Как уделить проекту больше времени

Сколько бы часов в день члены команды ни уделяли проекту, вы вряд ли откажетесь от возможности увеличить это число. Лучший, на мой взгляд, способ добиться этого придумал Франческо Чирилло из XP Labs. Он учит команды работать с высокой концентрацией 30-минутными интервалами (Cirillo, 2005). Каждый 30-минутный интервал состоит из двух частей: 25 минут интенсивной работы, а затем пятиминутный перерыв. Эти 30-минутные интервалы называют *pomodori*, что в переводе с итальянского означает «помидоры». Такое название связано с использованием таймеров в виде помидоров, подающих сигнал, когда истекают 25 минут.

Чирилло представил свой метод Пьергьюлиано Босси, который документально подтвердил его успешность при работе со многими командами (Bossi, 2003; Bossi and Cirillo, 2001). Эти команды обычно планировали выполнить работу на 10 помидоров (пять часов) в день. Если вы тратите свое время менее продуктивно, чем хотелось бы, можете попробовать этот подход.

Развертывание историй и выбор того, что подходит

Следующий этап — развертывание историй в задачи, их оценка и повторение этого процесса до тех пор, пока не удастся заполнить расчетное

количество доступных часов (в нашем случае 240 часов). Вовсе не обязательно разворачивать истории в порядке приоритета. Что нам действительно требуется, так это довольно случайный набор историй. Не старайтесь, например, развернуть все одно- и двухпунктовые истории и ни одной трех- и пятипунктовой истории. Аналогичным образом не старайтесь развернуть только истории, которые связаны в основном с пользовательским интерфейсом или базой данных. Попробуйте подобрать представительный набор историй.

Продолжайте отбирать истории и разбивать их на задачи до тех пор, пока выбранные задачи не превысят возможности членов команды. В случае команды SwimStats, например, следует соблюдать осторожность и рассчитывать на то, что программист и аналитик также являются опытными администраторами баз данных. Отбирайте истории до тех пор, пока команда с одним набором специальностей не сможет справиться с дополнительной работой. Подсчитайте количество пунктов или идеальных дней для выбранной работы, и это будет вероятная скорость команды.

Предположим, мы собрали запланированную команду (или создали ее аналог, если проект начнется лишь через год) и развернули некоторые истории, как показано в табл. 16.2. Если мы видим, что команда SwimStats в составе четырех человек может справиться с этим, но дополнительная работа будет ей не по силам, то останавливаемся. Эта работа объемом 221 час довольно хорошо подходит для доступных 240 часов. Наша точечная оценка скорости при этом будет равна 25.

Таблица 16.2. Часы и пункты для некоторых историй SwimStats

История	Пункты	Часы на задачи
Как тренер я могу вводить имена и гендерно-возрастную информацию по всем пловцам моей команды	3	24
Как тренер я могу определять расписание тренировок	5	45
Как пловец я могу видеть свое время во всех соревнованиях	2	18
Как пловец я могу корректировать свою гендерно-возрастную информацию	1	14
Как пловец я могу видеть линейную диаграмму своих результатов в конкретных заплывах	2	14
Как тренер я могу видеть линейную диаграмму прогресса в течение сезона для всех моих пловцов в конкретных заплывах	3	30
Как пловец я могу видеть секторную диаграмму, показывающую, сколько раз я занимал первое, второе и третье место	2	12
Как тренер я могу видеть текстовый отчет, показывающий лучшее время каждого пловца в каждом заплыве	2	14
Как тренер я могу загружать результаты соревнований из файла, экспортированного из системы учета времени на соревнованиях	5	50
Всего	25	221

Создание диапазона на основе найденной скорости

Используйте любой подход, который вам нравится, для преобразования точечной оценки скорости в диапазон. Как и прежде, я предпочитаю умножение на 60% и 160%. Для команды SwimStats наши расчетные 25 пунктов на итерацию дают диапазон от 15 до 40.

Возможный вариант для некоторых команд

Некоторым командам, особенно тем, в которых много членов с неполной занятостью, не следует закладывать одинаковое количество доступных часов для всех. В такие команды могут входить члены, которые могут уделять проекту намного меньшую часть своего времени. В этих случаях полезно создавать нечто вроде табл. 16.3.

В команде SwimStats, как видно из табл. 16.3, Юрий и Саша полностью заняты в проекте. SwimStats — единственный проект Сергея, однако он выполняет другие управленческие и корпоративные функции, которые отнимают у него определенное время. Карина делит свое время между SwimStats и другим проектом. У нее очень мало обязанностей, кроме этих двух проектов, поэтому она может уделять им почти шесть часов в день.

Вместе с тем ей приходится многократно переключаться с одного проекта на другой в течение дня, и такая многозадачность сказывается на ее производительности, поэтому в таблице показано, что она уделяет проекту SwimStats только два продуктивных часа в день.

Таблица 16.3. Расчет доступности для команды с неполной занятостью членов

Член команды	Доступные часы в день	Количество доступных часов на итерацию
Сергей	4	40
Юрий	6	60
Карина	2	20
Саша	6	60
Всего		180

Не забывайте, зачем это делается

Помните, что причиной, по которой мы прогнозируем скорость, является невозможность или нецелесообразность выполнения итерации при отсутствии исторических данных у конкретной команды. Такое может случиться, если команда пока что не сформирована, а вам нужно составить план проекта, который начнется лишь через несколько месяцев.

Если, например, вы работаете в условиях, когда стратегическое планирование и бюджетирование приходится осуществлять задолго до начала проекта, то прогнозирование скорости может оказаться наилучшим подходом.

Какой подход следует использовать

Принятие решения о том, какой подход использовать, нередко проще, чем может показаться при виде разнообразия вариантов. Обстоятельства чаще всего направляют вас и ограничивают возможности. Ниже приведены рекомендации по применению методов оценки скорости в порядке убывания их желательности.

- Если у вас есть возможность выполнить одну или несколько

итераций, прежде чем давать оценку скорости, то обязательно пользуйтесь ею. Никакая оценка не может сравниться с фактическими результатами, и наблюдение реальной скорости команды всегда будет наилучшим выбором.

- Используйте фактическую скорость команды в сходном проекте.
- Оценивайте скорость по тому объему работы, с которым вы можете справиться.

Независимо от подхода, который вы используете, при первой же возможности переходите на фактические, наблюдаемые значения скорости. Допустим, вы оцениваете скорость по объему работы, которая подходит для итерации, поскольку проект начнется не раньше чем через шесть месяцев и организации необходимо лишь примерно понять, сколько времени потребуется на этот проект. Как только начинается реализация проекта и у вас появляется возможность измерить фактическую скорость, переходите при обсуждении проекта и вероятного диапазона сроков его завершения на фактические данные.

Резюме

Существует три способа оценки скорости. Во-первых, можно использовать исторические средние показатели при их наличии. Вместе с тем, прежде чем применять исторические средние показатели, необходимо понять, не изменились ли существенно команда, характер проекта, технология и т.п. Во-вторых, можно отложить оценку скорости до тех пор, пока не будут выполнены несколько итераций. Это обычно наилучший вариант. В-третьих, можно спрогнозировать скорость путем разбивки нескольких историй на задачи и определения объема работ, подходящего для итерации. Этот процесс очень похож на планирование итерации.

Независимо от выбранного подхода оценки скорости следует представлять в виде диапазона, отражающего присущую оценке неопределенность. Конус неопределенности помогает определить подходящий размер диапазона.

Вопросы для обсуждения

1. В табл. 16.2 истории, оцениваемые одинаковым числом пунктов, имеют разную оценку задач в часах. Почему? (Если вам нужна подсказка, обратитесь к разделу «Соотнесение оценок задач с

пунктами» в главе 14 «Планирование итерации».)

2. Составьте таблицу, подобную табл. 16.3, для вашего текущего проекта. Существует ли возможность каким-либо образом увеличить количество времени, которое каждый член группы может посвящать проекту?

Глава 17

Буферизация планов для компенсации неопределенности

Неопределенность неудобна, определенность — глупа.

Китайская поговорка

Мне нередко жалуются на то, что agile-планирование не очень хорошо работает в некоторых ситуациях. При этом обычно называют следующие условия:

- Планирование проекта осуществляется задолго до его начала.
- Проект необходимо выполнить до жестко установленного срока и реализовать жестко заданный набор функций.
- Проект передается на договорной основе из одной организации в другую.
- Требования понятны только на очень поверхностном уровне.
- Организация противится слишком большой гибкости календарных графиков, даже в проектах без жестких сроков и четкого определения поставляемого продукта.

Умение составлять надежные планы в таких условиях чрезвычайно важно. Зачастую использования простого подхода, рассмотренного в предыдущих главах, недостаточно. В подобных условиях общим у всех проектов является то, что каждый из них характеризуется либо дополнительной неопределенностью, либо более серьезными последствиями в случае ошибки. Я, например, был когда-то вице-президентом по разработке программного обеспечения в компании, входящей в список Fortune 40, и календарный график для наших проектов обычно составлялся за 12–18 месяцев до их начала, когда мы верстали бюджет на следующий год. Мы, конечно, не фиксировали требования, однако должны были определять характер продукта, который будет создаваться. Несмотря на то, что представление о требованиях было расплывчатым, нам приходилось принимать обязательства первого уровня, которые позволяли составлять адекватное штатное расписание для

организации. Существовала и другая неопределенность: я редко знал задолго до начала проектов, кто именно из моей команды будет над ними работать. Чаще всего исполнителей просто не было в штате.

Сравните эту ситуацию с проектом, в котором от вас требуют установить жесткий срок и взять обязательство по реализации ключевого набора функций. Нарушение сроков или поставка неполной функциональности подорвет репутацию компании в отрасли и вашу репутацию в компании. Даже если вы достаточно хорошо понимаете требования и знаете, кто войдет в состав команды (в отличие от меня в приведенном выше примере), риск допустить ошибку очень значителен.

Для таких случаев характерна либо повышенная неопределенность, либо более серьезные последствия ошибок в графике релиза. Как результат, полезно создавать буфер при составлении календарного графика. Буфер — это допуск на ошибку в оценке. Когда значительна неопределенность или издержки, связанные с ошибкой, создание буфера очень разумно. Буфер помогает защитить проект от влияния неопределенности. Таким образом, буферизация календарного графика проекта становится хорошей стратегией управления риском. В этой главе мы рассмотрим два типа буферов: функциональный буфер и буфер календарного графика (временной буфер).

Функциональный буфер

Прошлым вечером я отправился в продовольственный магазин со списком продуктов из 37 пунктов. В моем распоряжении было лишь полчаса, чтобы купить нужные продукты и вернуться домой, поскольку я хотел посмотреть баскетбольный матч, который начался в это время. Как обычно, я начал с одного конца магазина и стал продвигаться к другому. Я шел вдоль полок, а мое внимание было сконцентрировано примерно на 20 товарах, которые нам требовались больше всего. Я знал: если приду домой без молока, хлеба или нарезки из индейки, то в перерыве мне придется снова идти в магазин. Если же я забуду что-то менее важное, то моя жена смирится с этим, у моих дочек будет что поесть, а у меня — возможность спокойно посмотреть баскетбол.

Буферизация проекта с помощью функций точно такой же процесс. Клиентам говорят: «Мы реализуем все функции в этом пакете, а в идеале некоторые функции в том пакете». В agile-проекте функциональный буфер создается очень просто. Сначала клиент выбирает всю абсолютно обязательную работу. Оценки этой работы суммируются. Данная работа представляет собой минимум, который может войти в релиз. Затем клиент выбирает еще 25–40% работ ближе к узкоспециализированному концу

диапазона для проектов с более высокой неопределенностью или меньшей устойчивостью к риску невыполнения календарного графика. Оценки этих работ добавляют к первоначальной оценке и получают суммарную оценку проекта. После этого составляют план проекта как обычно, предусматривая поставку *полного* набора функций, однако при этом часть работы является опциональной и выполняется только в том случае, если позволяет время. Опциональная работа выполняется в последнюю очередь и всегда после завершения обязательной.

Для лучшего понимания того, как это работает, предположим, что владелец продукта идентифицировал работу объемом 100 пунктов как обязательную. Каждая из отобранных историй предполагает выпуск продукта, который будет хорошо принят рынком. Владелец продукта затем выбирает дополнительные 30% работы, идентифицируя пользовательские истории, оцениваемые еще в 30 пунктов. Эти истории добавляются в проект как опциональная работа. Ожидаемый суммарный размер проекта теперь составляет 130 пунктов. Используя методы, описанные в главе 16 «Оценка скорости», команда определяет, что ее скорость составит 10 пунктов на однонедельную итерацию. В план проекта закладывают 13 итераций ($130 / 10$). Если все идет хорошо, то обязательная работа выполняется за первые 10 итераций, а оставшиеся три итерации посвящают опциональным функциям.

Процесс функциональной буферизации согласуется с agile-процессом, называемым «метод разработки динамических систем (dynamic systems development method — DSDM)». В DSDM-проектах требования разделяют на четыре категории: обязательные, желательные, опциональные, ненужные. К обязательным может быть отнесено не более 70% запланированных функций в проекте. Таким образом, в DSDM-проектах создается функциональный буфер, эквивалентный 30% срока проекта.

Временной буфер

Допустим, мне нужно отправиться в аэропорт и успеть на рейс в Италию. (Ну вот, сплеховал! Можно было придумать место и получше.) Для авиаперелетов характерно очень жесткое расписание. Самолет улетит в любом случае, со мной или без меня. В план перемещения от дома до соответствующей посадочной зоны в аэропорту мне необходимо заложить достаточно времени, чтобы точно успеть на самолет, но не столько, чтобы оказаться в аэропорту за три дня до вылета.

Я продумываю все этапы: дорога в аэропорт, поиск места на парковке для автомобиля, регистрация и сдача багажа, прохождение пункта досмотра. Я прикидываю, сколько времени мне потребуется на это, если все

пройдет хорошо, и решаю, что на все уйдет 70 минут. Иначе говоря, определяю, сколько это *должно занять*. На деле на всё про всё может уйти даже чуть меньше, но нет никакой гарантии, что не потребуется намного больше. На шоссе может случиться авария, парковка может оказаться забитой, у стойки регистрации может образоваться длинный хвост, как, впрочем, и у пункта досмотра. В общем, времени может потребоваться заметно больше. Планировать, что все пойдет наперекосяк во время одной и той же поездки, не стоит. Однако эта поездка важна для меня и мне не хочется опоздать на самолет, поэтому я должен добавить какой-то запас к 70 минутам, в которые можно уложиться, если все будет нормально.

Скажем, я решил отправиться в аэропорт за 100 минут до вылета самолета. Если все идет хорошо, у меня останется 30 минут, чтобы побродить по аэропорту, а это не самое плохое занятие. Если обстоятельства сложатся совсем удачно (на шоссе я попаду в зеленую волну, на парковке найду место в первом ряду, и никого не будет у стойки регистрации или в пункте досмотра), то в аэропорту в моем распоряжении, возможно, окажется целых 40 свободных минут. Но если я застряну в пробке или долго простою в очереди на регистрацию, дополнительного времени, скорее всего, хватит, чтобы вовремя попасть на самолет. Дополнительные 30 минут — это мой буфер в календарном графике, который защищает своевременное завершение проекта в целом (достижение аэропорта).

В случае поездки в аэропорт у меня нет возможности прикидывать темп продвижения (скорость) и периодически предоставлять авиакомпании (клиенту) обновленные данные об ожидаемом времени моего прибытия. Это время и темп моего продвижения не интересуют авиакомпанию. Время вылета зафиксировано, как и сроки во многих проектах по разработке программного обеспечения. В таких ситуациях временной буфер служит защитой от неопределенности, которая может влиять на своевременное завершение проекта.

Обратите внимание на то, что меня не волнует, какой именно вид деятельности (поездка на автомобиле, парковка, регистрация или прохождение досмотра) займет слишком много времени. Меня заботит лишь, не уйдет ли слишком много времени на цепочку действий в целом. Чтобы успеть на самолет, я добавляю 30-минутный буфер ко всему календарному графику поездки в аэропорт. Аналогичным образом хотелось бы добавлять буфер в календарный график проектов с высоким уровнем неопределенности или с серьезными последствиями в случае невыполнения срока.

Отражение неопределенности в оценках

Для защиты календарного графика проекта от неопределенности необходимо количественно оценить эту неопределенность. При определении и присвоении однозначной оценки пользовательской истории мы претендуем на то, что отдельно взятое число отражает наши ожидания в отношении количества времени, которое потребуется для реализации функции. Более реалистично, однако, считать, что работа может быть завершена в пределах диапазона сроков. Команда может оценить конкретную пользовательскую историю в три идеальных дня, зная, что эти три идеальных дня обычно представляют четыре или пять дней работы. Если на реализацию истории потребуется шесть дней, то никто этому не удивится — работа иногда занимает больше времени, чем планировалось. Если представить графически возможные сроки выполнения задачи, то получим примерно такую кривую, которая показана на рис. 17.1.

Кривая имеет такую общую форму потому, что обычно мало что можно сделать для ускорения выполнения задачи, однако существует бесконечное множество вещей, которые могут пойти не так и задержать ее выполнение. Например, когда я уже почти заканчиваю кодирование конкретной новой функции, мой компьютер зависает и несохраненные изменения теряются. В наше здание попадает молния и уничтожает хранилище данных. Мы направляем запрос на поставку резервной копии к следующему утру, но служба доставки теряет носитель. Мне, однако, уже все равно, поскольку по пути на работу меня переезжает пресловутый автобус.

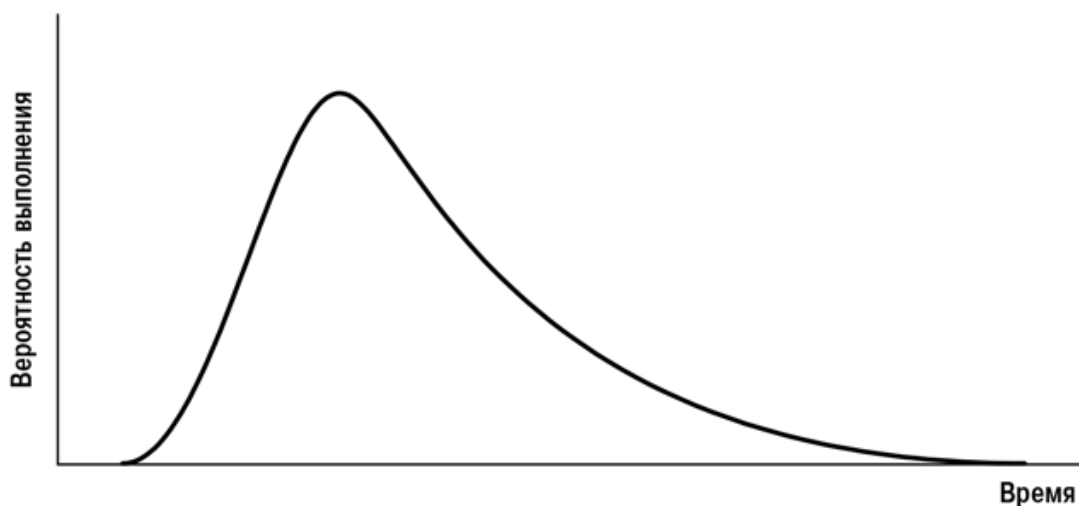


Рис. 17.1. Распределение времени выполнения задачи

Наиболее вероятное время выполнения задачи на рис. 17.1 находится в том месте, где кривая достигает пика. В целом, однако, вероятность выполнения задачи к этому сроку составляет менее 50%. Это известно потому, что слева от пика находится менее 50% площади под кривой. Если бы разработчик дал оценку, соответствующую пику на рис. 17.1, он, скорее

всего, не выполнил бы работу к названному сроку. Более наглядно это представлено на рис. 17.2, где изображена кумулятивная вероятность завершения задачи в сроки, отложенные по горизонтальной оси, или раньше них.

Если на рис. 17.1 показана вероятность завершения задачи в конкретное время, то на рис. 17.2 представлена вероятность завершения задачи в это время или ранее. В процессе оценки и планирования это более важно для нас, чем вероятность завершения работы в отдельно взятый день (как показано на рис. 17.1).

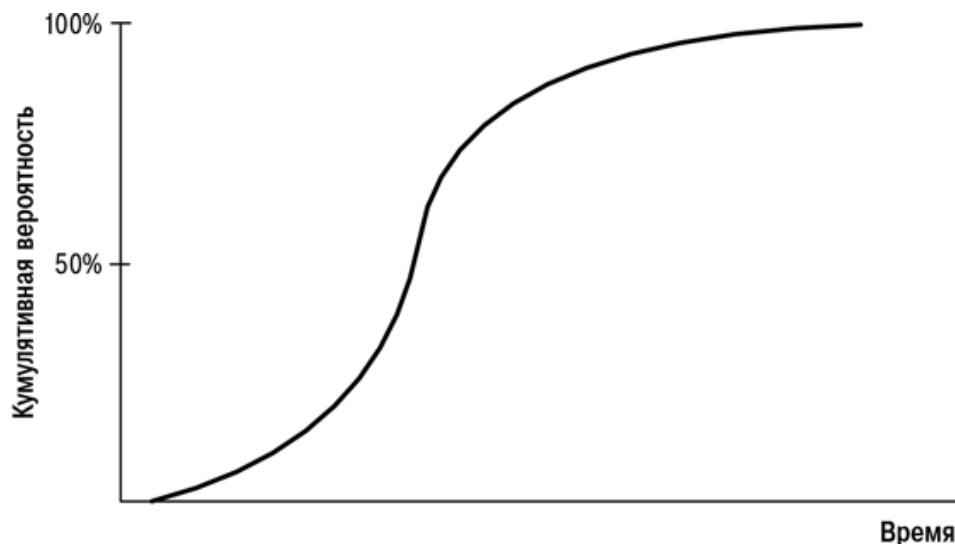


Рис. 17.2. Кумулятивное распределение времени завершения задачи

Посмотреть на рис. 17.2 и кумулятивную вероятность времени завершения задачи можно и иным образом. Представьте, что 100 одинаково квалифицированных и опытных программистов независимо разрабатывают новую функцию. К какой дате каждый из них завершит работу? Результаты могут быть аналогичными тем, что приведены в табл. 17.1. В этой таблице показано число завершивших работу в каждый из дней и, что более важно, суммарное число завершивших работу в определенный день.

Таблица 17.1. Количество разработчиков, завершивших реализацию функции к определенному дню

День	Число завершивших работу в этот день	Суммарное число завершивших работу
День 1	5	5
День 2	40	45
День 3	25	70
День 4	15	85
День 5	10	95
День 6	5	100

Допустим, нам нужна 90%-ная уверенность в выполнении календарного графика. Одним из путей достижения этого является оценка срока реализации каждой пользовательской истории с 90%-ной вероятностью и использование полученных оценок. Однако если сделать это, то календарный график проекта почти наверняка окажется слишком растянутым. Чтобы посмотреть, как работает временной буфер, вернемся к моей поездке в аэропорт, вероятный календарный график которой показан на рис. 17.3.

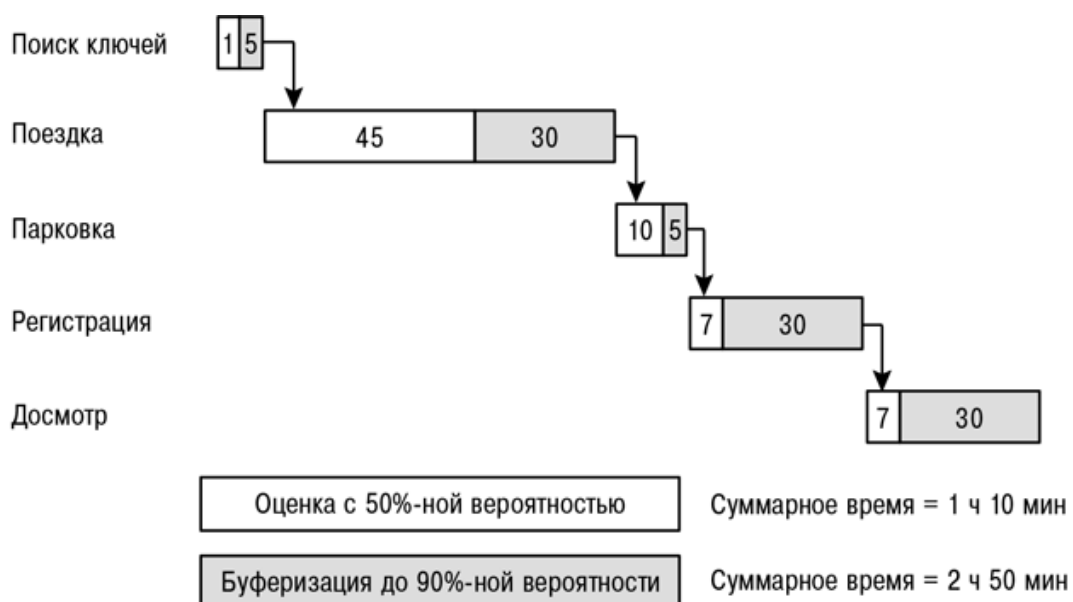


Рис. 17.3. Оценки продолжительности действий, позволяющие попасть на определенный авиарейс с вероятностью 50% и 90%

Первое число для каждой задачи (в светлом прямоугольнике) — оценка

продолжительности выполнения задачи с 50%-ной вероятностью. По моим предположениям, одна половина задач потребует больше времени, а другая — меньше. Второе число (в затемненном прямоугольнике) — это дополнительное время, необходимое для достижения 90%-ной вероятности. Дополнительное время, представляющее собой разницу между 50%-ной и 90%-процентной оценками, называют *локальным запасом*. Мы нередко добавляем локальный запас к оценке, когда хотим быть более уверенными в выполнении задачи. В нашем случае я думаю, что на поиски ключей мне может понадобиться от одной до шести минут. Я могу добраться до аэропорта через 45–75 минут.

Сложение оценок с 50%-ной вероятностью дает мне ожидаемый срок, равный одному часу и 10 минутам. Однако, если я выйду из дома так близко ко времени вылета, малейшая задержка приведет к опозданию на рейс. Вместе с тем сложение оценок с 90%-ной вероятностью дает в сумме два часа 50 минут. Мне совершенно не хочется выезжать почти за три часа до вылета, поскольку вероятность того, что все пойдет наперекосяк, очень мала. Что мне реально хочется получить, так это план проекта, подобный приведенному на рис. 17.4.

План на рис. 17.4 строится на основе оценок с 50%-ной вероятностью, к которым прибавляется проектный буфер. Такой план намного разумнее, чем тот план, который полностью построен на суммировании оценок с 50%-ной или 90%-ной вероятностью. В плане на рис. 17.4 защищается только конечный срок, который имеет принципиальное значение: общий срок проекта. Поскольку совершенно не важно, потребует ли та или иная задача на пути в аэропорт больше времени, мне не нужно создавать буфер для обеспечения своевременного завершения задач. Это позволяет мне сконструировать календарный график, где нет локального запаса для отдельных задач, зато часть этого запаса добавляется в буфер, защищающий календарный график в целом. Обратите внимание на то, что календарный график с буфером на рис. 17.4 занимает только один час 58 минут — почти на час меньше, чем график, полученный в результате суммирования оценок с 90%-ной вероятностью.

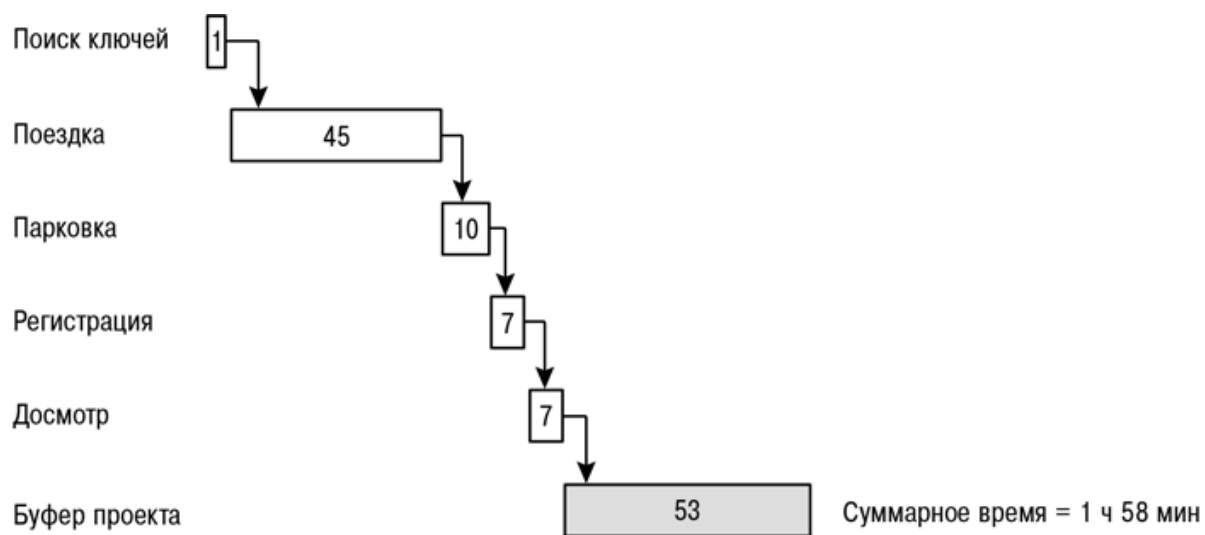


Рис. 17.4. Буферизованная поездка в аэропорт

Еще лучше то, что перенос локального запаса в буфер проекта в целом позволяет избежать влияния закона Паркинсона и синдрома студента. Как мы знаем из главы 2 «Почему планирование дает неудовлетворительные результаты», закон Паркинсона гласит, что работа растягивается так, чтобы занять все отведенное на нее время. Синдром студента (Goldratt, 1997) — это склонность откладывать дела на последний момент, что мешает их успешному завершению, например начинать работу над курсовым проектом в колледже за три дня до срока сдачи. Устраняя проблемы, связанные с законом Паркинсона и синдромом студента, такой подход делает вероятность выполнения более короткого календарного графика с буфером выше, чем вероятность выполнения более длинного графика.

Первое, что требуется для создания временного буфера, это пересмотр нашего процесса оценки таким образом, чтобы обеспечить генерирование двух оценок для каждой пользовательской истории или функции. Точно так же, как и в случае поездки в аэропорт, нам нужны оценки с 50%-ной и 90%-ной вероятностью. Получить их довольно просто: когда команда соберется для проведения оценки, начните с 50%-ной оценки первой истории. Затем дайте 90%-ную оценку этой истории, прежде чем переходить к следующей истории или функции.

Определение размера буфера проекта

Размер буфера проекта на рис. 17.4 определен как 53 минуты. Как я получил его? На основе 50%-ных и 90%-ных оценок для этого проекта, как показано на рис. 17.4. Прелесть присвоения двух оценок каждой пользовательской истории в том, что цифры предельно ясно указывают на уровень риска срыва графика, связанный с каждым элементом. Например, если одну историю оценивают в 3–5, а вторую в 3–10, то мы знаем, что вторая история

привносит в проект более значительный риск срыва графика. Буфер проекта должен иметь такой размер, чтобы компенсировать риск срыва графика, связанный с запланированной работой. Возьмем экстремальный вариант: если вы составляете календарный график проекта, то вам потребуется меньший буфер проекта для задач с оценкой 3–7 по сравнению с задачами, которые имеют оценку 3–100. Если проект включает в себя трехпунктовые задачи, который могут обернуться 100-пунктовыми, то проекту нужен более значительный буфер, чем при задачах, которые в наихудшем случае могут потянуть только на 10 пунктов. Иначе говоря, спред между 50%-ными и 90%-ными оценками влияет на размер буфера проекта [7].

Поскольку наши оценки представляют собой пункты для каждого элемента при вероятности 50% и 90%, разница между ними равна примерно двум стандартным отклонениям. Стандартное отклонение для каждого элемента равно $(w_i - a_i) / 2$, где w_i представляет наихудший случай (90%-ную оценку) для истории i , а a_i — средний случай (50%-ную оценку) для той же истории. Нам нужно, чтобы буфер проекта защищал проект в целом на таком же 90%-ном уровне, как и у каждой задачи с ее 90%-ной оценкой. Это означает, что наш буфер проекта должен составлять два стандартных отклонения, которые определяются по следующей формуле:

$$2\sigma = 2 \cdot \sqrt{((w_1 - a_1)/2)^2 + ((w_2 - a_2)/2)^2 + \dots + ((w_n - a_n)/2)^2},$$

где σ — стандартное отклонение. Эту формулу можно упростить до такого вида:

$$2\sigma = \sqrt{(w_1 - a_1)^2 + (w_2 - a_2)^2 + \dots + (w_n - a_n)^2}.$$

Посмотрим, как эта формула применяется для определения размера временного буфера. Предположим, что наш проект включает в себя шесть пользовательских историй, представленных в табл. 17.2, и что каждая история имеет 50%-ную и 90%-ную оценки. Оценки могут быть как в пунктах, так и в идеальных днях. В последней колонке табл. 17.2 приведен результат расчета, в котором из наихудшей (90%-ной) оценки истории вычитают среднюю (50%-ную) оценку этой истории, а разность возводят в квадрат. Для первой истории, например, результат составляет $(3 - 1)^2 = 4$. Временной буфер — это квадратный корень из суммы этих квадратов [8]. Временной буфер в нашем случае равен квадратному корню из 90, или 9,4, которые мы округляем до 9. Общая продолжительность проекта представляет собой сумму 50%-ной оценки и буфера проекта, в нашем

случае $17 + 9 = 26$.

Таблица 17.2. Расчет буфера для проекта с шестью историями

Пользовательская история	Средний вариант (50%-ный)	Худший вариант (90%-ный)	(Худшее — среднее) ²
Как и любой другой посетитель сайта, я хотел бы видеть показатели каждого пловца	5	13	64
Как и любому другому посетителю сайта, мне необходимо пройти процедуру идентификации, прежде чем я получу доступ к чувствительным разделам	1	3	4
Как пловец я хочу видеть расписание тренировок	3	5	4
Как пловец или родитель я хочу знать, где расположены бассейны спортивной лиги	5	8	9
Как и любой другой посетитель сайта, я хочу видеть национальные рекорды по возрастным группам и по соревнованиям	2	5	9
Как и любой другой посетитель сайта, я хочу видеть результаты по всем соревнованиям	1	1	0
Всего	17	35	90

Буфер, рассчитанный в табл. 17.2, интуитивно понятен. Наибольший вклад в размер буфера вносит та пользовательская история (первая история), которая имеет наибольшую неопределенность (разница в восемь пунктов между 50%-ной и 90%-ной оценками). В то же время история, где неопределенность отсутствует (последняя история), не добавляет в буфер ничего.

Создание буфера для календарного графика в одних случаях приводит к увеличению длины проекта на одну или несколько итераций, а в других — нет. Чаще всего приводит. Допустим, команда в нашем примере спрогнозировала свою скорость как девять пунктов на итерацию. Если проект был оценен в 17 пунктов (сумма 50%-ных оценок), то следовало бы ожидать завершения проекта за две итерации. Однако при включении в проект буфера полный объем работы становится равным 26 пунктам, для реализации которых потребуются три итерации при скорости команды, равной девяти пунктам.

Более простой способ расчета буфера

Предыдущий подход к определению размера буфера проекта является

наилучшим, однако бывает, что вы не можете получить 50%-ную и 90%-ную оценки. На этот случай существует упрощенный определения размера буфера. Оцените каждую историю на 50%-ном уровне, а затем примите за размер буфера половину суммы 50%-ных оценок. Убедитесь в том, что все члены команды знают о необходимости давать оценку с 50%-ным уровнем уверенности. Нам нужны оценки, которые с одинаковой вероятностью могут оказаться как завышенными, так и заниженными.

Хотя такой расчет значительно проще, у него есть серьезный недостаток — в нем не учитывается фактическая неопределенность конкретных пользовательских историй в проекте. Допустим, у нас есть две истории, каждая из которых оценена в пять пунктов. Обе они вносят одинаковый вклад в буфер проекта (половину своего размера, или по 2,5 пункта). Это правило сохраняется несмотря на то, что одна из этих историй может иметь 90%-ную оценку 100, а другая — 10.

Правила определения размера буфера

Независимо от того, что вы предпочитаете, метод квадратного корня из суммы квадратов или 50%-ный подход, при определении размера буфера необходимо руководствоваться следующими дополнительными правилами, вытекающими из рекомендаций Лича (Leach, 2000).

- Метод квадратного корня из суммы квадратов является самым надежным, если оцениваются не менее 10 пользовательских историй или функций. Если же в вашем проекте меньше 10 элементов, то, скорее всего, можно обойтись без буфера.
- На буфер проекта должно приходиться не менее 20% общего срока этого проекта. Буфер меньшего размера не всегда обеспечивает адекватную защиту проекта в целом.

Комбинирование буферов

На первый взгляд наличие нескольких буферов может показаться перебором. Вместе с тем нередко вполне уместно использовать несколько буферов, которые защищают проект от разных типов неопределенности. В автомобиле есть и ремни, и подушки безопасности, поскольку они защищают от разных типов столкновений. Тот или иной тип неопределенности проекта всегда следует компенсировать соответствующими средствами, а это означает, что неопределенность функций нужно компенсировать с помощью функционального буфера, а неопределенность календарного графика — с помощью временного. Кроме

того, при наличии нескольких буферов размер каждого из них может быть меньше.

Именно сочетание функционального и временного буферов обеспечивает реальную защиту проектов от неопределенности. Взгляните на три проекта, показанные на рис. 17.5. В секции *a*) представлен проект, где необходимо поставить заказчику четко определенный набор функций, но можно варьировать сроки. В секции *b*) представлена противоположная ситуация: проект, где дата завершения фиксирована, но допускается гибкость в отношении реализованных функций. В секции *c*) видно, что создание и функционального, и временного буфера позволяет команде выдержать дату поставки и одновременно реализовать минимальный набор функций. При создании плана релиза наша цель заключается в использовании буферов таким образом, чтобы команда могла принять эти виды обязательств.



Рис. 17.5. Три проекта с разными подходами к буферизации

Не забывайте также, что в проекте могут использоваться и другие буферы, помимо функционального и временного. В проект можно включить *бюджетный буфер*, где, например, на проект выделяются 30 разработчиков, а средства позволяют увеличить их число до 33. Это обычная практика для средних и крупных проектов, однако в небольших проектах к ней прибегают не так часто по двум причинам.

1. Один-два дополнительных человека для создания буфера по персоналу в небольшом проекте почти всегда вносят прямой вклад в срок проекта. Увеличение штата с 30 до 33 разработчиков вряд ли заметно повысит производительность, если вообще повысит. Если же штат увеличивается с четырех разработчиков до пяти, то производительность практически наверняка вырастет.
2. Очень трудно создавать буферы для чего-то небольшого. Когда в проекте, где занято 30 человек, создается резерв из трех работников и полный штат увеличивается до 33, мы получаем 10%-ный буфер по

персоналу. Аналогичный буфер для проекта, в котором заняты три человека, составит всего 0,3 разработчика. Понятно, что часть человека не добавишь в проект.

Временной буфер — это не раздувание сроков

Под *раздуванием* понимают произвольное добавление времени к оценке и относятся к этому явлению негативно. Оценка раздувается, когда я считаю, что на работу нужно три дня, но на всякий случай говорю «пять». Люди раздувают оценку, если ожидают неприятностей в случае ошибки. Буфер для календарного графика к этой категории не относится: временной буфер — это необходимая маржа безопасности, добавляемая к сумме оценок, из которых удален локальный запас.

Вы держите дистанцию, равную пяти длинам автомобиля, между своей и впередиидущей машиной, потому что этот буфер может потребоваться при резком торможении. Может, конечно, случиться, что вам удастся продержаться час-другой на расстоянии одной длины автомобиля до впередиидущей машины, но это маловероятно. Буферная дистанция вокруг автомобиля критически важна для вашей безопасности. Аналогичным образом буферы вокруг вашего проекта критически важны для его безопасности.

Небольшая гибкость даты поставки и функциональности позволяет нам буферизировать два измерения проекта. Еще важнее то, что мы создаем буферы из соответствующих ресурсов под ограничения каждого проекта: для срока проекта создается временной буфер, для функциональности — функциональный. Когда не удастся буферизировать ограничения должным образом, приходится увеличивать размер других буферов. Если меня вынуждают гарантировать функциональность, то я поддерживаю гарантию, увеличивая временной буфер.

Ограничительные оговорки

Хотя понимание того, как создать один или несколько буферов в проекте, очень важно, полезно также помнить о некоторых ограничениях, связанных с использованием буферов.

- При добавлении временного буфера используйте подход с двумя оценками, описанный в этой главе, а при использовании одной оценки следите за тем, чтобы она была 50%-ной. Добавление временного буфера к и без того пессимистичным 90%-ным оценкам

приведет к чрезмерному удлинению сроков.

- Во многих проектах конкретный срок с точным набором поставляемой функциональности не требуется. Вместо этого команде просто нужно поставить высококачественное программное обеспечение как можно быстрее в пределах определенного периода. В этом случае не обременяйте себя дополнительной работой по созданию буферов для проекта.
- Не скрывайте информацию о буферах. Не следует скрывать их наличие или предназначение. Не забывайте, что буфер (особенно временной) может показаться раздуванием сроков. Это означает, что необходимо информировать заинтересованные стороны о том, как вы получили эти оценки и буфер и как этот буфер повышает уверенность в выдерживании сроков календарного графика.

Резюме

Большинство проектов отличаются очень высокой неопределенностью. Эта неопределенность зачастую не полностью отражается в календарном графике и в сроках, устанавливаемых проектной командой. Иногда неопределенность настолько высока или значительна, что при оценке продолжительности проекта для ее учета необходимо принимать дополнительные меры. Это особенно характерно для случаев, когда проект планируют задолго до начала работ, когда нужно уложиться точно в срок, реализовав относительно жесткий набор функций, когда проект передается сторонней организации, когда требования определены лишь поверхностно или когда неправильное определение срока влечет за собой значительные последствия (финансовые или иные).

Наиболее распространенными являются функциональные и временные буферы. Функциональный буфер создается, когда требования к продукту приоритизированы и признано, что реализовать можно не все функции. В DSDM-процессе, например, рекомендуется считать 30% работ опциональными, создающими функциональный буфер для проекта. Когда время истекает, календарный график все равно можно выдержать, отказываясь от реализации элементов функционального буфера.

Временной буфер, в свою очередь, создается путем включения в календарный график дополнительного времени с учетом неопределенности, присущей размеру команды. Функциональный буфер можно сконструировать на основе присвоения каждой пользовательской истории двух оценок — с 50%-ной и 90%-ной вероятностью. Формула «извлечение квадратного корня из суммы квадратов» позволяет найти подходящий

размер временного буфера.

Проект следует защищать от неопределенности функций с помощью функционального буфера и от неопределенности календарного графика с помощью временного буфера. Функциональный буфер можно комбинировать с временным буфером. Обычно это дает хороший результат, поскольку позволяет уменьшить размер обоих буферов.

Вопросы для обсуждения

1. Дает ли затрата дополнительных усилий на расчет временного буфера какие-либо выгоды в условиях вашей организации?
2. Предусмотрены ли в вашем текущем проекте буферы для компенсации какого-либо типа неопределенности? Если да, то какие? Если нет, то какие типы буферов были бы наиболее полезными?
3. Есть ли признаки проявления закона Паркинсона или синдрома студента в вашей организации? Что еще, помимо предложений, представленных в этой главе, вы можете предпринять для снижения влияния этого закона и синдрома?

Глава 18

Планирование проекта с участием нескольких команд

Занимайтесь планированием, а о планах забудьте.

Мэри Поппендик

Нередко говорят, что agile-команды состоят не более чем из 7–10 разработчиков. Команды такого размера могут выполнить довольно большой объем работы, особенно при использовании agile-процесса, который открывает возможности для повышения производительности и поощряет ее повышение. Вместе с тем встречаются проекты, к которым хотелось бы привлечь более крупную команду. В таких случаях вместо создания команды из 100 человек agile-подход предполагает создание нескольких небольших команд. В agile-проекте может участвовать десяток небольших команд вместо одной огромной команды из 100 человек.

В этой главе мы возьмем то, что узнали в предыдущих главах, и применим к задаче планирования проекта, в котором участвуют несколько команд. Для планирования крупного проекта с участием нескольких команд необходимо:

1. Принять общую базу для оценок.
2. Быстрее добавлять детали в пользовательские истории.
3. Выполнять опережающую разработку планов.
4. Включать в план поддерживающие буферы.

При осуществлении проекта может потребоваться применение некоторых или всех этих приемов в зависимости от количества участвующих команд, а также от того, насколько часто и интенсивно им придется координировать свою деятельность. Как правило, я рекомендую командам использовать столько дополнительных приемов, сколько необходимо, начиная с установления общей базы и т.д., вплоть до создания поддерживающих буферов.

Принятие общей базы для оценок

Хотя было бы хорошо предоставить каждой команде возможность по своему усмотрению выбирать, в чем проводить оценку — в пунктах или идеальных днях, в большинстве случаев проекты с участием нескольких команд выигрывают от оценки в одних и тех же единицах и установления их базового смысла. Представьте, как трудно было бы предсказывать, сколько еще времени потребуется для реализации набора пользовательских историй, если бы одна их часть оценивалась в идеальных днях, а другая — в пунктах. Хуже того, представьте, насколько все усложнилось бы, если бы одна команда оценила набор историй в 20 пунктов, а другая — в 100 пунктов.

В начале проекта команды должны встретиться и определить, что они будут использовать, пункты или идеальные дни. Затем им нужно установить общую базу для оценок с тем, чтобы оценка одной команды была идентична оценке другой команды, если бы эта работа попала к ней. Каждая пользовательская история оценивается только одной командой, но оценки должны быть эквивалентными, независимо от того, какая команда оценивает работу.

Существует два способа определения общей базы. Один из них эффективен, только если команды работали вместе в предыдущем проекте. В этом случае они могут выбрать несколько пользовательских историй из прошлого проекта и принять их оценки за ориентир. Допустим, команды используют для оценки идеальные дни, тогда им нужно выбрать пару-тройку историй, каждая из которых оценивается в один идеальный день. Затем нужно найти несколько историй, которые оцениваются в два идеальных дня, и т.д. Таким образом надо подобрать порядка двух десятков старых историй и договориться о новой оценке каждой из них. После согласования таких базовых историй команды могут самостоятельно оценивать другие истории, сравнивая их с базовыми (т.е. выполняя оценку по аналогии).

Второй подход похож на первый, однако предполагает коллективную оценку разнообразных новых пользовательских историй. При этом выбирают различные истории, которые планируется включить в новый релиз. Эти истории должны иметь разные размеры и относиться к таким областям системы, с которыми их связывают большинство оценщиков. Либо вся большая команда, либо представители каждой из небольших команд, когда команда в целом слишком велика, собираются и согласуют оценки этих историй. Как и в первом случае, эти оценки затем становятся базовыми и используются для сравнительной оценки будущих историй.

Отдельные команды могут рассматривать оценку в разных единицах без общей базы, только если создаваемые продукты реально не связаны друг с

другом и абсолютно исключена возможность перехода разработчиков из одной команды в другую. Так или иначе, даже в такой ситуации я рекомендую устанавливать общую базу, поскольку это упрощает обмен информацией по проекту.

Более быстрое добавление деталей в пользовательские истории

В идеальном случае agile-команда начинает итерацию с нечетко определенными требованиями и к концу итерации превращает эти требования в функционирующую, протестированную программу. Пройти путь от нечетких требований к работающей программе за одну итерацию обычно легче в проекте с участием одной команды. Когда команд несколько, зачастую очень полезно и даже необходимо включить больше комментариев в пользовательские истории перед началом итерации. Дополнительные детали облегчают координирование работы команд.

С этой целью в крупных командах нередко выделяют специальных аналитиков, дизайнеров пользовательских интерфейсов и других специалистов, которые тратят часть своего времени в процессе выполнения текущей итерации на подготовительную работу к следующей итерации. Как правило, я не рекомендую заставлять аналитиков, дизайнеров пользовательских интерфейсов и других специалистов заранее полностью прорабатывать итерацию. Их задача по-прежнему заключается в выполнении работы, связанной с текущей итерацией, однако при планировании этой итерации они должны включать в план некоторые задачи, связанные с подготовкой к следующей.

Что я считаю самым полезным результатом работы, выполняемой до начала итерации, так это идентификацию условий удовлетворенности владельца продукта для пользовательских историй, которые с наибольшей вероятностью попадут в следующую итерацию. Одним из условий удовлетворенности владельца продукта для пользовательских историй является приемочное тестирование высокого уровня пользовательской истории, прежде чем она будет сочтена завершенной. Пользовательская история считается завершенной, когда может быть продемонстрировано выполнение всех условий удовлетворенности, идентифицированных владельцем продукта.

Несмотря на то, что знать условия удовлетворенности для пользовательской истории до начала итерации очень полезно, маловероятно (да и не нужно), чтобы команда заблаговременно идентифицировала их для всех пользовательских историй. На практике точный набор историй,

которые войдут в следующую итерацию, неизвестен вплоть до конца совещания по планированию итерации, которое проводится непосредственно перед началом этой итерации. В большинстве случаев, однако, владелец продукта и команда могут сделать правдоподобное предположение относительно историй, которые с наибольшей вероятностью войдут в следующую итерацию. Именно для них имеет смысл идентифицировать условия удовлетворенности до начала итерации.

Опережающее планирование

Большинство команд с умеренно сложным или частым взаимодействием выигрывают от создания скользящего перспективного окна во время планирования релиза и итерации. Предположим, что две команды работают над приложением SwimStats. В определенной мере приложение связано с отображением статичной информации, такой как время тренировки, адреса и маршруты подъезда к бассейнам. Однако на SwimStats должна также отображаться динамическая информация из базы данных, в том числе результаты всех соревнований за последние 15 лет и индивидуальные достижения всех пловцов во всех заплывах.

Информация по национальным рекордам и рекордам по возрастным группам хранится в базе данных на удаленном объекте национальной ассоциации по плаванию. Доступ к этой базе данных не так прост, как хотелось бы командам, и национальная ассоциация собирается сменить поставщиков баз данных в ближайший год-два. По этой причине владелец продукта и команды разработчиков согласились с тем, что нужно разработать API (интерфейс прикладного программирования) для доступа к базе данных. Это должно значительно упростить переход к другому поставщику баз данных. Первоначальные пользовательские истории и их оценки приведены в табл. 18.1.

Таблица 18.1. Первоначальные пользовательские истории и оценки для SwimStats

Пользовательская история	Пункты
Как SwimStats мы хотим иметь возможность легко отказаться от нашего поставщика баз данных	30
Как и любому другому посетителю сайта, мне необходимо пройти процедуру идентификации, прежде чем я получу доступ к чувствительным разделам	20
Как пловец я хочу видеть расписание тренировок	10
Как пловец или родитель я хочу знать, где расположены бассейны спортивной лиги	10
Как и любой другой посетитель сайта, я хочу видеть национальные рекорды по возрастным группам и по соревнованиям	10
Как и любой другой посетитель сайта, я хочу видеть результаты по всем соревнованиям	10
Как и любой другой посетитель сайта, я хочу видеть результаты каждого пловца	20

Скорость оценивается как 20 пунктов на итерацию для одной и другой команды. Поскольку объем работ составляет 110 пунктов, команды должны поставить полную функциональность за три итерации. Вместе с тем 30 пунктов приходится на разработку API, а еще 40 пунктов (три последние истории в табл. 18.1) можно реализовать только после разработки API. Это приводит к такому распределению работ, которое представлено на рис. 18.1, где взаимозависимость команд обозначена стрелкой между реализацией API первой командой и работой над индивидуальными результатами, выполняемой второй командой.

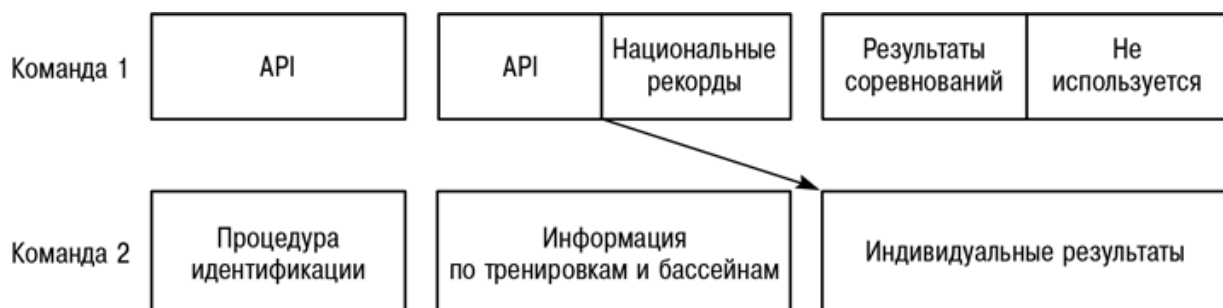


Рис. 18.1. Координация работы двух команд

Возможно, вы помните, что в главе 13 «Основные аспекты планирования релиза» я рекомендовал показывать в плане релиза детали только следующих двух итераций. Объясняется это тем, что такой подход нередко достаточен для поддержки взаимозависимостей, с которыми имеют дело многие команды. Когда нескольким командам приходится работать вместе, план релиза следует обновлять с тем, чтобы показывать и координировать

работу в следующих двух или трех итерациях. Точное число итераций, конечно, зависит от частоты и серьезности взаимодействий команд. После завершения итераций связанные с ними детали удаляют из плана. Таким образом, план релиза становится *скользящим опережающим планом*, который всегда отражает ожидания относительно нескольких новых итераций. Лауфер (Laufer, 1996) называет это «заглядывание вперед».

На рис. 18.1 показана ситуация, в которой команды обмениваются информацией между итерациями. Это менее рискованно, чем планирование на основе информации, передаваемой во время осуществления итерации. В начале очередной итерации каждая команда идентифицирует работу, которую она может выполнить, и принимает обязательства по ее выполнению. В случае, представленном на рис. 18.1, в начале третьей итерации команда 2 может принять полноценное обязательство по реализации пользовательской истории, связанной с данными по индивидуальным результатам, поскольку она знает о завершении API. Предположим вместо этого, что команда 2 планирует свою третью итерацию, когда API еще не разработан, а завершение работы над ним ожидается через несколько дней. Даже если бы команда 2 могла завершить историю, связанную с данными по индивидуальным результатам, без API в первый же день, ее обязательство было бы более шатким, и весь календарный график подвергся бы значительному риску. Командам нередко приходится принимать обязательства на основе результатов середины итерации. Тем не менее в той мере, в какой это возможно, им следует ограничиваться обязательствами, связанными с работами, которые завершены до начала очередной итерации.

Включение в план поддерживающих буферов

Для большинства команд в большинстве ситуаций скользящий опережающий план вполне уместен. Однако бывает, что взаимодействия команд настолько сложны или часты, что простого скользящего опережающего планирования, описанного в предыдущем разделе, недостаточно. В таких случаях первое, что нужно сделать, это попытаться сократить количество взаимодействий до уровня, при котором скользящее опережающее планирование станет реальным. Если добиться этого не удастся, попробуйте включить в итерацию *поддерживающий буфер*, который даст свободу, необходимую другим командам. Поддерживающий буфер, как и временной буфер, описанный в предыдущей главе, защищает своевременную поставку набора новых возможностей. Это довольно сложный способ сказать простую вещь: если вашей команде требуется что-

либо от моей команды к 8:00, то моей команде не следует планировать завершение этого на 7:59. Иначе говоря, нам нужен план вроде того, что представлен на рис. 18.2.



Рис. 18.2. Добавление поддерживающего буфера

На рис. 18.2 поддерживающий буфер вставлен между завершением API командой 1 и началом работы команды 2 с использованием API над пользовательской историей, связанной с данными по индивидуальным результатам. Поддерживающий буфер защищает дату начала работы над историей, связанной с данными по индивидуальным результатам, от задержек в реализации API.

Чтобы включить поддерживающий буфер в план релиза, нужно всего лишь временно планировать работу с расчетом на более низкую скорость команды, которая поставляет определенную возможность другой команде. В случае, представленном на рис. 18.2, усилия команды 1 равномерно делятся между завершением API и поддерживающим буфером так, чтобы гарантировать команде 2 возможность приступить к работе над индивидуальными результатами с самого начала третьей итерации. Это не означает, что команда 1 получает возможность не торопиться в процессе второй итерации. В действительности от нее ожидают, что она выполнит связанную с API работу в 10 пунктов, использует только часть буфера и начнет работать над пользовательской историей, связанной с национальными рекордами, уже во второй итерации.

Что именно буферизируется?

В ситуации, показанной на рис. 18.2, добавление поддерживающего буфера не увеличивает длину проекта в целом, поскольку третья итерация команды 1 была заполнена лишь частично. Во многих случаях, однако, добавление поддерживающего буфера увеличивает ожидаемую длину проекта. Правда, обычно это делают аккуратно, чтобы остаться в рамках реалистичных ожиданий в отношении вероятного календарного графика и не создать впечатление «раздувания графика в стремлении не особенно напрягаться».

Из-за того что поддерживающие буферы могут удлинять сроки, их следует добавлять только при необходимости.

Чтобы определить, где необходимы поддерживающие буферы, сначала распределите пользовательские истории между командами и итерациями. Затем идентифицируйте критические взаимозависимости между итерациями и командами. Наконец, создайте поддерживающий буфер между этими критическими взаимозависимостями. Иначе говоря, добавляйте поддерживающий буфер только тогда, когда команда не может выполнить запланированную высокоприоритетную работу без получения результатов работы другой команды. Если же команда может легко переключиться на другую работу с высокой ценностью, то поддерживающий буфер не нужен. Аналогичным образом не добавляйте поддерживающий буфер, если вторая команда может успешно продвигаться вперед, опираясь лишь на часть результатов работы первой команды. Когда ваша команда может начать работу, получив от моей команды всего половину запланированной функциональности, поддерживающий буфер не нужен.

Определение размера поддерживающего буфера

При определении размера поддерживающего буфера можно руководствоваться правилами, изложенными в главе 17 «Буферизация планов для компенсации неопределенности». Впрочем, к счастью, большинство межкомандных взаимозависимостей обычно связано не более чем с несколькими историями или функциями. Как результат, чаще всего у вас не хватает историй для эффективного использования метода «квадратный корень из суммы квадратов», описанного в этой главе. В таких случаях размер поддерживающего буфера принимается равным некоторому проценту от размера историй, в которых проявляется взаимозависимость. По умолчанию размер буфера устанавливается на уровне 50%, однако эту величину необходимо скорректировать с учетом оценки команды.

В принципе, размер поддерживающего буфера может превышать длину итерации, однако буфер такого размера редко когда целесообразен. Поддерживающий буфер, превышающий размер итерации, обычно является результатом намерения передать разработку значительной части функциональности другой команде. По двум причинам в этих случаях проекту почти наверняка не нужен поддерживающий буфер. Во-первых, при передаче работы другой команде ее всегда стараются разделить так, чтобы функциональность поставлялась постепенно. Это позволяет второй команде начать работу сразу же после получения исходного набора функций от первой. Во-вторых, вместо использования чрезвычайно большого поддерживающего буфера командам следует искать способы

перераспределения или внесения корректив в состав итераций при первых же признаках отставания. Владелец продукта или клиент поставляющей команды обычно позволяют двум командам определить такую последовательность поставки, которая устраивает всех.

В моей практике не было случаев использования поддерживающих буферов, размер которых превышал целую итерацию, однако изредка встречались буферы больше половины итерации. Рассматривая перспективу создания большого поддерживающего буфера, я анализирую принятые допущения и план и пытаюсь найти возможности сокращения цепочки результатов, передаваемых от одной команды к другой.

Но ведь это уйма работы

Конечно, но никуда не деться, когда имеешь дело с большим проектом, в котором участвует несколько команд. Не забывайте, что ничего такого не требуется, если у вас всего одна команда. Возможно, ничего такого не потребуется, даже когда у вас три или четыре команды по семь человек, если эти команды часто обмениваются информацией.

Вместе с тем во многих крупных проектах приходится устанавливать сроки и принимать обязательства по их исполнению за много месяцев до начала работ, и во многих крупных проектах существует взаимозависимость между командами, подобная той, что описана в этой главе. При реализации такого проекта полезно уделить планированию несколько дополнительных часов. Это позволит вам сразу более уверенно и точно оценить целевую дату завершения работ, а также создать определенную защиту от легко устранимых задержек.

Резюме

В крупных agile-проектах обычно стремятся избегать создания больших команд и вместо этого задействуют несколько команд. Когда несколько команд работают над одним проектом, им нужно координировать свою работу. В этой главе описаны четыре приема, помогающие командам работать над одним и тем же проектом.

Во-первых, команды должны установить общую базу для своих оценок. Командам необходимо договориться о проведении оценки в одних и тех же единицах: пунктах или идеальных днях. Также они должны договориться о значении этих единиц, согласовав оценки для небольшого набора историй.

Во-вторых, когда командам приходится работать вместе, нередко

полезно более быстро добавлять детали в пользовательские истории. Лучше всего использовать с этой целью идентификацию условий удовлетворенности владельца продукта для истории. Под ними понимаются аспекты, выполнение которых может быть продемонстрировано после реализации пользовательской истории.

В-третьих, команды выигрывают от создания скользящего опережающего плана в процессе планирования релиза. Скользящий опережающий план — это просто взгляд вперед на несколько будущих итераций (обычно на две-три), который позволяет командам координировать работу, обмениваясь информацией о том, над чем каждая из них начнет работать в ближайшем будущем.

В-четвертых, в очень сложных проектах с большим числом межкомандных взаимозависимостей в план полезно встраивать поддерживающие буферы. Поддерживающий буфер — это определенный запас по времени, который предотвращает задержку поставки результатов одной из команд, приводящую к задержке начала работ у другой команды.

Эти приемы обычно применяют в проектах в том порядке, в котором они описаны в этой главе. При необходимости, однако, их можно применять в любом порядке.

Вопросы для обсуждения

1. Как вы устанавливаете общую базу для оценок в проектах с участием нескольких команд?
2. Насколько значительна взаимозависимость команд в вашем текущем проекте? Какие приемы из описанных в этой главе являются самыми действенными?

Часть V

Отслеживание прогресса и информирование

В части IV этой книги мы говорили о том, как составить обоснованный календарный график для проекта. Процесс планирования составлением плана и календарного графика не завершается. Очень важным моментом является отслеживание прогресса относительно плана, информирование о прогрессе и уточнение плана на основе наблюдений.

В первых двух главах этой части мы разберем, как следить сначала за выполнением плана релиза, а потом за выполнением плана итерации. Настоящая часть завершается обзором способов информирования об оценках, планах и прогрессе. Помимо этого приводится образец итогового отчета в конце итерации.

Глава 19

Мониторинг плана релиза

Звезды могут лгать, а цифры — никогда.

Мэри Чапин Карпентер, песня «I Feel Lucky»

Перед древними моряками стояли две задачи. Во-первых, им нужно было знать, на какой широте они находятся, т.е. их положение на карте в направлении север–юг. Во-вторых, им нужно было знать долготу, т.е. положение в направлении восток–запад. Определение широты на основе наблюдений за Полярной звездой было сравнительно простым делом и практиковалось уже за 300 лет до Рождества Христова. С долготой было сложнее, поскольку для ее определения требовались относительно точные часы, или хронометр. К сожалению, достаточно точные хронометры (в частности, такие, которые годились бы для использования на борту корабля) появились лишь в начале XVIII в.

До изобретения хронометра моряк мог лишь примерно оценить, на какой долготе он находится. Такие оценки строились на последовательных расчетах и поправках, которые называются *навигационным счислением пути*. Счисление пути предполагает примерную оценку того, как далеко на восток или запад уплыл корабль, и внесение поправок с учетом примерной оценки влияния ветров и морских течений. Допустим, измеритель скорости на вашей яхте показывает, что вы делаете восемь миль в час. Однако если вы идете против волн, ветра и течения, то фактическое продвижение может составить всего пять миль в час. Навигационное счисление пути должно отражать все эти факторы, иначе вы оцените долготу неправильно.

Отслеживание прогресса команды разработчиков программного обеспечения очень похоже на определение положения корабля, особенно путем навигационного счисления пути. Нам хотелось бы направить процесс разработки программного обеспечения по прямой линии из точки А в точку В. Такое, однако, удастся редко в силу изменения или уточнения требований, отклонения фактической скорости продвижения от ожидаемой, а также ошибок, допускаемых при определении нашего положения. В этой главе рассматриваются методы отслеживания прогресса, позволяющие минимизировать влияние этих или подобных проблем.

Отслеживание процесса разработки релиза

Перед началом работы над релизом мы составляем план, в котором говорится что-то вроде: «В течение следующих четырех месяцев и восьми двухнедельных итераций мы выполним работу объемом примерно 240 пунктов (или идеальных дней)». По мере того как мы узнаем больше об истинных потребностях пользователей и о размере проекта, эта оценка может меняться. Вместе с тем мы должны в любой точке иметь возможность оценить наше положение относительно цели — выполнения определенного объема работы за определенное время.

При такой оценке необходимо учитывать множество действующих факторов. Первое, и в идеале самое главное, — прогресс, которого добивается команда. Второе — изменение объема проекта. Владелец продукта может добавлять или удалять требования. Если он добавляет 40 пунктов, а команда реализует 30, то у нее остается больше работы, чем в начале предыдущей итерации. Цель сдвигается, и полезно знать, как далеко команда находится от достижения новой цели. Или разработчики могут получить во время итерации такие знания, которые заставят их пересмотреть оценки в пунктах, присвоенные последующим работам в плане релиза.

Эти факторы (выполненная работа, изменение требований и пересмотренные оценки) можно считать аналогами силы ветра (называемой *дрейфом*) и силы моря (называемой *сносом*). Посмотрите на рис. 19.1, где показаны силы, действующие на парусную лодку. Лодка на этом рисунке проходит меньшее расстояние, чем то, которое должно получиться на основе показаний ее измерителя скорости. Аналогичным образом, даже если компас лодки показывает на восток в течение всего плавания, ветер заставит эту лодку сместиться к югу. Без корректировки курса нашей лодке потребуется больше времени, чтобы добраться до цели, не вполне совпадающей с первоначальной. Мысленно переименуйте стрелки на рис. 19.1 так, чтобы дрейф и снос стали изменением требований (добавлением или удалением функций) и изменением оценок. Теперь рис. 19.1 отражает проблемы отслеживания прогресса программного проекта по отношению к его календарному графику.

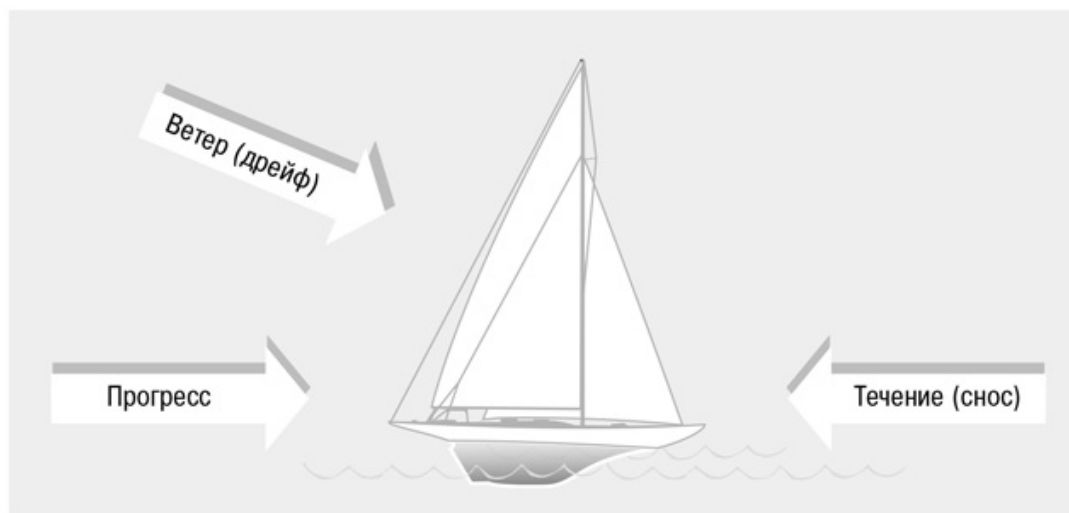


Рис. 19.1. Силы, действующие на парусную лодку

Скорость

Темп продвижения корабля измеряют в узлах; agile-команда измеряет темп своего продвижения в единицах скорости. Скорость выражается как число пунктов (или идеальных дней), реализованных за итерацию. О команде, которая реализовала 12 историй, оцениваемых в сумме в 30 пунктов, в прошлой итерации, говорят, что ее скорость равна 30, или 30 пунктам на итерацию. Если считать, что объем проекта не меняется, то именно на столько меньше работы осталось выполнить команде.

Скорость — это основной показатель прогресса команды, поэтому очень важно установить базовые правила ее расчета. Прежде всего команда должна учитывать пункты при определении скорости только для тех историй или функций, которые полностью завершены к концу итерации. Завершение — это не что-то вроде «кодирование завершено, но программа еще не протестирована» или «программа написана, но ее нужно еще интегрировать». Под завершенной понимают программу, которая хорошо написана, хорошо сбалансирована, проверена и вычищена, а кроме этого удовлетворяет стандартам кодирования и прошла все тесты. Чтобы узнать, какой прогресс был достигнут, мы учитываем пункты только той работы, которая завершена.

С подсчетом объема незавершенной работы связаны три проблемы. Во-первых, незаконченную, или незавершенную, работу чрезвычайно трудно измерить. Что считать более завершенным — пользовательскую историю, которая была запрограммирована, но не протестирована, или историю, которая частично запрограммирована и частично протестирована? Насколько далеко продвинулся программист, который спроектировал решение для истории, но не начал кодирование? Мы хорошо знаем объем того, что не было начато, и довольно хорошо знаем объем того, что уже

выполнено. Оценивать объем работы необходимо в одном из этих состояний и ограничиться этим.

Во-вторых, незавершенные истории разрушают доверие между командой-разработчиком и командой-клиентом в проекте. Если историю не удастся завершить во время итерации, как планировалось, разработчики и клиент должны совместно решать проблему, как только она обнаруживается. Обычно это означает, что историю удаляют из итерации или разбивают ее на части и некоторые из них удаляют. Владелец продукта и команда-клиент могут принимать такие решения в режиме реального времени в процессе итерации и изменять приоритеты на основе новых знаний о стоимости истории. Как вариант, команда-клиент может изменить критерии приемки истории и сделать их более мягкими. Она не должна, конечно, опускаться до приемки неотлаженной или непротестированной версии истории, но вполне может снизить требования к производительности в конкретных ситуациях и т.п.

В-третьих, и это самое важное, незаконченная работа ведет к увеличению объема незавершенного производства в процессе разработки. Чем больший объем незавершенного производства допускает команда, тем больше времени требуется для превращения новых функций из сырых идей в функционирующее программное обеспечение. С течением времени это снижает производительность команды в целом. Аналогичным образом при значительном объеме незавершенного производства в процессе команде нужно больше времени на получение обратной связи по разрабатываемому продукту. Это означает задержку обучения.

Если у команды остаются незавершенные истории в конце итерации, то она работает с функциями или историями слишком большого размера. Небольшие истории обеспечивают стабильный поток работы на всем протяжении процесса разработки. Если истории остаются незавершенными, их необходимо разбивать на части. Вместе с тем если в процессе выполнения итерации выясняется, что та или иная история оказалась длиннее, чем ожидалось, то к этому необходимо привлечь внимание владельца продукта. Владелец продукта должен совместно с командой найти способ разбить историю или сократить ее объем так, чтобы можно было завершить эту часть в пределах текущей итерации, а остаток перенести в будущую итерацию.

Так как же команде учесть частично завершенную историю при оценке скорости? Как она будет оценивать такую историю, менее важно, чем определение причин, по которым это произошло, и того, как избежать повторения подобной ситуации. Причиной могла быть недооценка истории. В этом случае команде необходимо выяснить, какой тип работы был недооценен или забыт, и попытаться не забывать о нем при проведении оценки в будущем. А может быть, история оказалась незавершенной из-за

того, что в текущую итерацию включили слишком много историй. В таком случае необходимо более тщательно планировать итерацию.

Диаграмма выгорания релиза

На рис. 19.2 приведена диаграмма выгорания релиза (Schwaber and Beedle, 2002). На вертикальной оси откладывается количество оставшихся пунктов в проекте. (С таким же успехом это может быть количество оставшихся идеальных дней.) На горизонтальной оси откладываются итерации. Диаграмма выгорания релиза показывает количество оставшейся работы в начале каждой итерации. Она служит эффективным визуальным индикатором скорости продвижения команды к ее цели. На рис. 19.2 представлена гипотетическая диаграмма выгорания для проекта объемом 240 пунктов, реализуемых равными частями за восемь итераций.

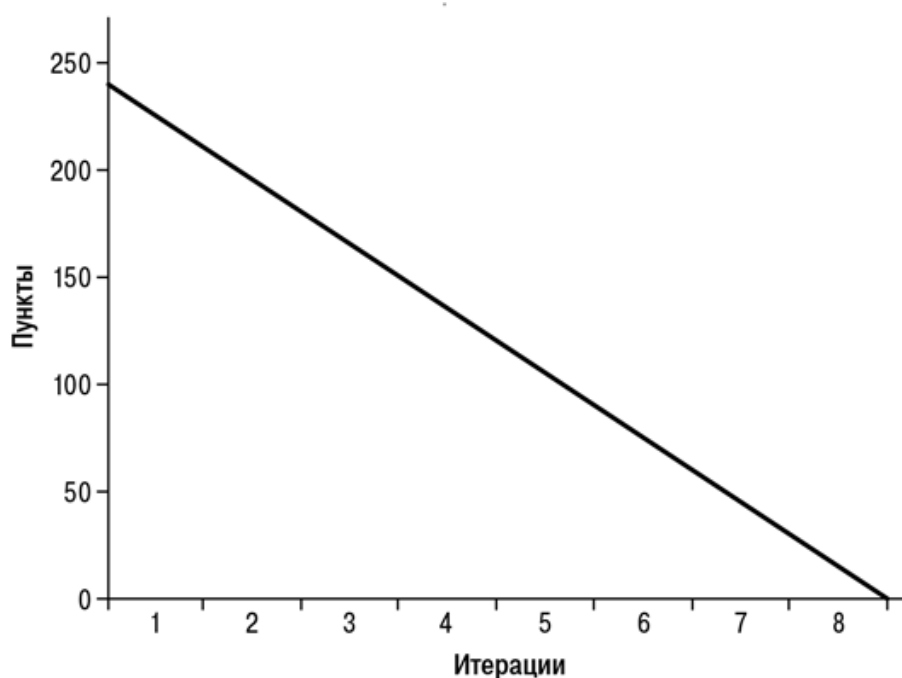


Рис. 19.2. 240-пунктовый проект, реализуемый за восемь итераций

Конечно, маловероятно, что команда, которая имеет ожидаемую скорость 30, будет иметь в точности такую скорость в каждой итерации. Скорее всего, диаграмма выгорания 240-пунктового релиза будет выглядеть так, как показано на рис. 19.3.

На рис. 19.3 представлен прогресс команды после трех итераций. Ее прогресс нестабилен. В первой итерации она выполнила объем работы, довольно близкий к плановым 30 пунктам. Однако в конце второй итерации у нее фактически осталось больше работы, чем было выполнено к началу

этой итерации. Такая ситуация могла возникнуть, например, если команда выяснила, что разработка пользовательского интерфейса требует намного больше усилий, чем предполагалось первоначально, и увеличила свои оценки для всех оставшихся историй, связанных с пользовательскими интерфейсами.

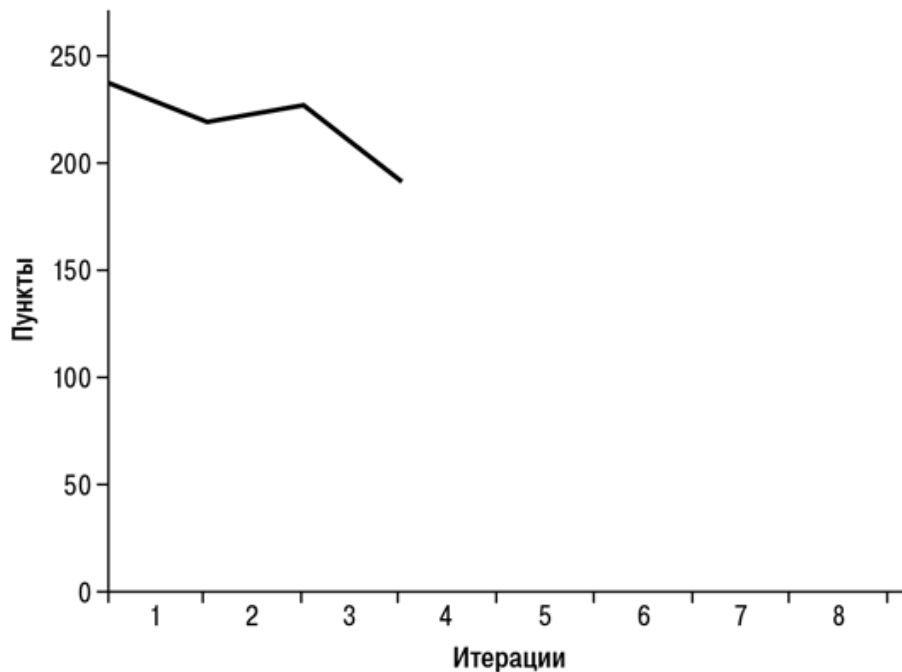


Рис. 19.3. Более реалистичная диаграмма выгорания для 240-пунктового проекта после третьей итерации

Как вариант, диаграмма может демонстрировать не выгорание, а возрастание в результате того, что в релиз добавилась работа. Аналогом этого является парусная лодка, делающая 8 миль в час, которая попадает в течение противоположного направления, имеющее скорость 12 миль в час. В результате лодка оказывается дальше от своей первоначальной цели. Вместе с тем в софтверном проекте дополнительная работа может быть результатом приобретения командой знания, которое направляет ее к созданию более ценного релиза.

Чтобы понять, как это происходит, предположим, что во второй итерации команда опять выполнила работу объемом 30 пунктов, но владелец продукта идентифицировал еще 40 пунктов работы как необходимые для релиза. В этом случае чистым результатом будет более значительный объем работы в конце второй итерации, чем в ее начале. Поскольку диаграмма выгорания отражает чистый прогресс команды, на рисунке мы видим поворот линии вверх.

Вы можете спросить, почему мы строим диаграмму именно так. Это связано с тем, что именно при таком представлении одна диаграмма просто и ясно показывает два самых важных показателя, которые можно

использовать для отслеживания проекта: объем оставшейся работы и чистый темп продвижения команды, не зависящий от изменений объема проекта. Представьте, что вы член команды, чей прогресс представлен на рис. 19.3. В конце третьей итерации вас спрашивают, будет ли релиз завершен за плановые восемь итераций. А если не будет, то у вас запросят более точную оценку предполагаемого срока завершения. Вы можете ответить на этот вопрос, просто взглянув на диаграмму выгорания на рис. 19.3. Проведите прямую линию через 240 пунктов на вертикальной оси и количество пунктов, оставшихся в проекте на текущий момент. Там, где прямая линия пересечет горизонтальную ось, и будет ожидаемый срок завершения проекта. С первого взгляда на рис. 19.3 понятно, что проект не удастся завершить за плановые восемь итераций.

Гистограмма выгорания релиза

С одной стороны, диаграмма выгорания релиза, показанная на рис. 19.3, предельно эффективна. Она понятна, и ее можно быстро объяснить любому в организации. Диаграмма такого вида очень информативна и показывает нам, когда проект будет завершен, если факторы, влияющие на него, останутся неизменными. С другой стороны, иногда полезно представить диаграмму выгорания так, чтобы сделать очевидными изменения скорости команды и объема работы. Для этого нужно построить гистограмму выгорания релиза вроде той, что показана на рис. 19.4.

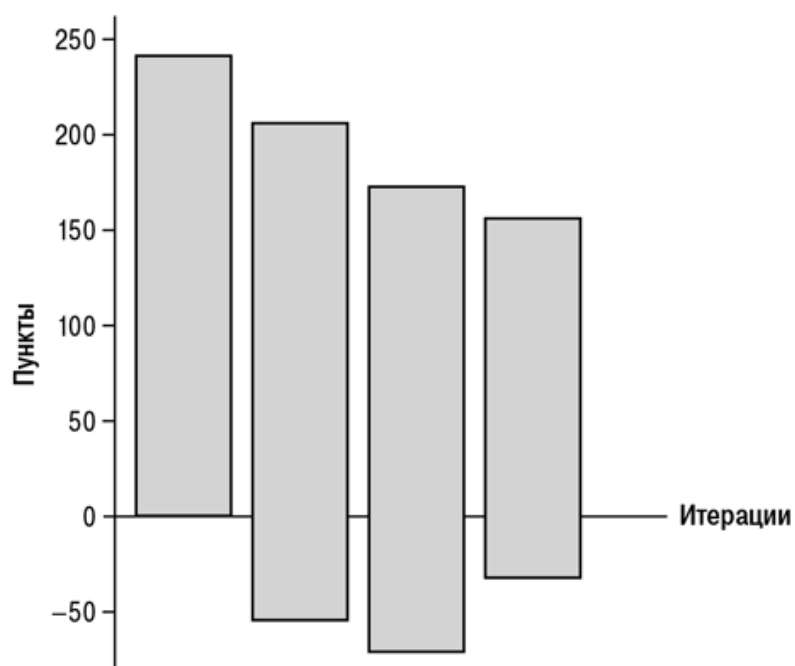


Рис. 19.4. Разделение влияния изменений скорости и объема

Каждый столбик на рис. 19.4 показывает объем работы в релизе на

начало итерации. В этом типе диаграммы выгорания используются столбики вместо линий, что позволяет различать области выше и ниже горизонтальной оси на уровне нуля. Нижняя часть столбика опускается вниз, когда в проект добавляют работу, и смещается вверх, когда работу удаляют из итерации. Если нижняя часть столбика опустилась ниже горизонтальной оси, значит в релизе увеличился общий объем работы.

Для того чтобы понять, как это работает, приведу пример. На рис. 19.4 по плану релиз должен включать объем работы, равный 240 пунктам. В начале первой итерации на диаграмме выгорания находится один столбик, вытянутый от 0 до 240. Как и прежде, команда ожидает, что ее скорость будет равна 30, а работу удастся завершить за восемь итераций. В первой итерации команда достигает ожидаемой скорости, и поэтому верхняя часть второго столбика находится на уровне 210. Владелец продукта, однако, решил, что в релизе должно быть больше функций, чем считалось первоначально. Работу, которую решили добавить к релизу, оценили в 50 пунктов. По этой причине столбик второй итерации опустился ниже нулевой линии и стал занимать пространство от -50 до 210. Другими словами, объем работы в релизе теперь составляет 260 пунктов, т.е. больше, чем вначале. К концу второй итерации, глядя на рис. 19.4, можно отметить три интересных факта.

1. Скорость команды находится на ожидаемом уровне. Это видно по выгоранию, о котором говорит расположение вершин первых двух столбиков.
2. Был добавлен значительный объем работы. Это видно по положению нижней части второго столбика. Предположительно работа была добавлена потому, что она должна привести к созданию более ценного релиза. Вместе с тем полезно обратить внимание на темп изменения объема этого проекта — к нему было добавлено больше, чем завершено. Возможно, это не такой уж тревожный факт, все зависит от того, сохранится ли тренд и насколько важна первоначальная целевая дата завершения релиза.
3. Суммарный объем работы, оставшейся в релизе, больше ее объема в начале проекта. Это следует из того, что общая высота второго столбика больше высоты первого столбика.

Очевидно, что данное представление диаграммы выгорания значительно более наглядно. Его недостаток заключается в том, что смысл диаграммы не ясен сразу же.

Посмотрим теперь на вторую и третью итерации проекта на рис. 19.4. Во второй итерации команда вновь достигла целевой скорости. Владелец продукта опять добавил работу, однако на этот раз прибавка не такая

большая, как в предыдущей итерации.

Что произошло в третьей итерации? При ее выполнении скорость снизилась до 20. Это может быть результатом недооценки некоторых историй, включенных в итерацию, болезни или ухода в отпуск какого-нибудь члена команды либо переоценки части оставшихся работ. Команда могла выполнить плановый объем работы 30 пунктов, однако повышение оценок некоторых оставшихся историй привело к тому, что чистый прогресс составил 20 пунктов вместо 30.

Однако самый интересный момент, связанный с третьей итерацией, показан в нижней части четвертого столбика. Во время этой итерации владелец продукта удалил функцию из релиза. При выпуске релиза проект по-прежнему содержит больше пунктов, чем было запланировано первоначально. Такой вывод следует из того, что столбик все еще находится ниже нулевого уровня. Вместе с тем на этом этапе проект содержит меньше запланированных пунктов, чем было в начале предыдущей итерации. При этом не имеет значения, какие функции удалялись — те, что были изначально в плане релиза, или те, что были добавлены владельцем продукта в предыдущих итерациях. Приоритизация работы — прерогатива владельца продукта, который может добавлять и удалять функции по своему усмотрению. Чистый эффект показан в нижней части столбика выгорания.

При построении диаграммы выгорания такого типа нужно помнить четыре простых правила.

- Каждый раз, когда работа завершается, верхняя часть столбика понижается.
- Когда работу переоценивают, верхняя часть столбика смещается вверх или вниз.
- Когда добавляют новую работу, нижняя часть столбика смещается вниз.
- Когда работу удаляют, нижняя часть столбика смещается вверх.

Посмотрим на гистограмму выгорания релиза реального проекта, которая представлена на рис. 19.5. Здесь мы видим, что команда демонстрирует хороший прогресс в течение первых двух итераций. Владелец продукта добавляет небольшой объем работы перед началом второй итерации, что довольно часто случается. Во время третьей итерации команда обнаруживает, что некоторые пользовательские истории недооценены, и проводит переоценку части оставшейся работы. Это приводит к повышению верхней части четвертого столбика на рис. 19.5. Перед началом четвертой итерации владелец продукта удаляет работу из плана релиза. Как результат, нижняя часть столбика смещается вверх выше нулевой линии. Во время этой итерации команда демонстрирует хороший прогресс. Далее план релиза больше не меняется, и команда продвигается

вперед вплоть до завершения работы.

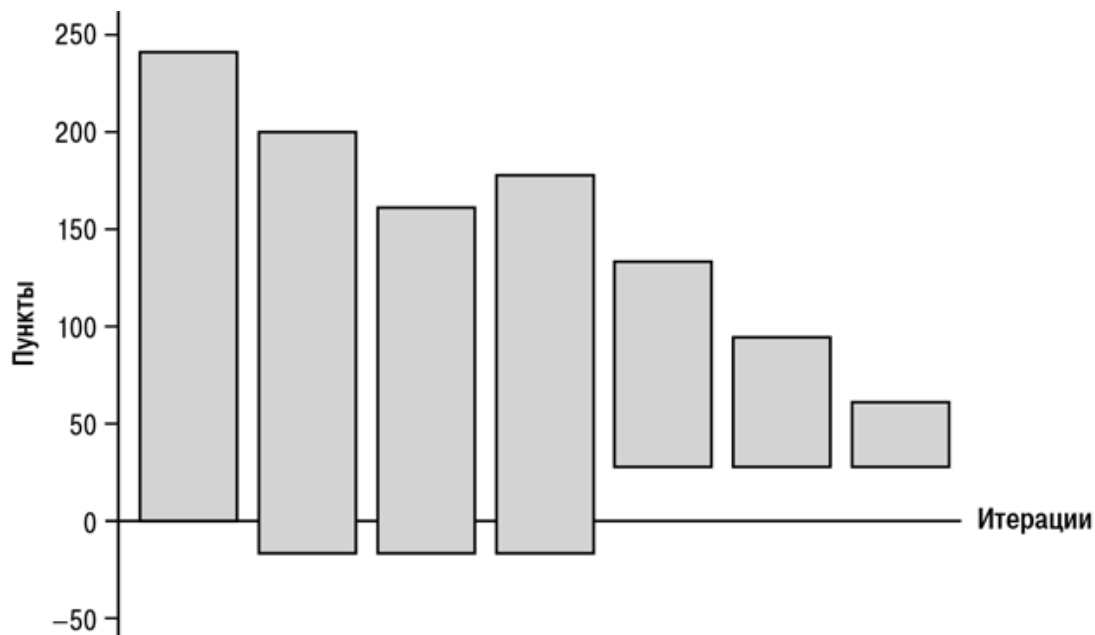


Рис. 19.5. Удаление работы из релиза

Особенности применения гистограммы выгорания релиза

Хотя я считаю, что гистограмма выгорания релиза более наглядна (и, следовательно, зачастую более полезна), чем традиционная диаграмма выгорания, существует пара ограничений ее использования. Во-первых, гистограмму выгорания труднее понимать, поэтому в новых командах я всегда начинаю работу с более простой традиционной диаграммы. Во-вторых, гистограмма выгорания предназначена только для тех организаций, которые достигли достаточной зрелости и могут удержаться от споров о том, что находится выше линии, а что ниже. При первых же признаках спора такого характера я говорю всем участникам проекта, что мы не можем использовать гистограмму и возвращаемся к традиционной диаграмме выгорания.

Диаграмма парковки

Джефф Делюка (Deluca, 2002) предложил еще один способ визуализации процесса реализации командой запланированных функций релиза. На рис. 19.6 показан вариант того, что Делюка называет диаграммой парковки. *Диаграмма парковки* содержит большой прямоугольник для каждой темы (или группы пользовательских историй) в релизе. В каждом прямоугольнике указано название темы, количество историй в этой теме,

количество пунктов или идеальных дней для историй и процент реализованных пунктов.



Рис. 19.6. Диаграмма парковки показывает, в какой мере реализована каждая тема

В прямоугольнике «Гендерно-возрастная информация пловцов» на рис. 19.6 видно, что эта тема включает в себя восемь историй, которые оцениваются в сумме в 36 пунктов. Из этого числа уже выполнено 18 пунктов, поскольку, как мы знаем, реализовано 50% функции. Мы не можем сказать, какие именно пользовательские истории реализованы. Отдельные прямоугольники на диаграмме можно даже выделить цветами, чтобы показать, реализована ли тема, идет ли ее реализация в соответствии с календарным графиком, требует ли эта тема повышенного внимания или значительно отстает от календарного графика.

Диаграмма парковки — эффективный способ отражения большого объема информации на небольшом пространстве. Во многих случаях диаграмма парковки позволяет обобщенно представить все темы релиза на одном листе.

Резюме

Скорость — это показатель объема работы, выполняемой командой за каждую итерацию. Скорость должна определяться на основе правила «все или ничего». Если история реализована, то команда может включить ее полную оценку в расчет скорости. Если история частично реализована в течение итерации, то она не учитывается при определении скорости.

Диаграмма выгорания релиза показывает количество пунктов или идеальных дней, остающихся в проекте на начало каждой итерации. Прогресс команды никогда не бывает равномерным. Он варьирует, в частности, в результате неточности оценок, изменения оценок и изменения объема работы. Диаграмма выгорания может отражать также увеличение

объема работы в течение итерации. Это происходит в тех случаях, когда команда, хотя она и выполняет определенную работу, обнаруживает, что оценки оставшейся работы были заниженными, или когда увеличивается объем проекта. Ключом к интерпретации диаграммы выгорания релиза является понимание того, что она отражает чистый прогресс команды, т.е. ее прогресс за вычетом работы, добавленной в релиз.

Гистограмма выгорания релиза представляет собой полезный в некоторых случаях вариант стандартной диаграммы выгорания. Гистограмма позволяет разделить прогресс команды относительно запланированной работы и изменения объема релиза. Изменения объема релиза отображаются как смещение столбика ниже горизонтальной оси. Этот тип диаграммы более нагляден, чем стандартная диаграмма выгорания, однако его следует использовать осмотрительно, поскольку он может вызвать в организации споры относительно того, на какую часть столбика повлияли изменения — на верхнюю или на нижнюю.

Диаграмма парковки полезна для получения общего представления о прогрессе команды при реализации тем, включенных в план проекта.

Вопросы для обсуждения

1. Если вы не используете диаграмму выгорания релиза в своем текущем проекте, какой результат дало бы построение такой диаграммы в конце каждой итерации?
2. Какой из методов мониторинга и представления прогресса, описанных в настоящей главе, был бы наиболее результативным для вашего текущего проекта?
3. Какие заинтересованные стороны в вашем проекте не получают информацию, которая была бы полезной для них?

Глава 20

Мониторинг плана итерации

Факты лучше, чем мечты.

Уинстон Черчилль

В дополнение к отслеживанию прогресса в направлении общей цели релиза всегда полезно отслеживать прогресс команды разработчиков в выполнении работы отдельной итерации. В этой главе мы рассмотрим два инструмента отслеживания процесса выполнения работ в пределах итерации: доску задач и диаграмму выгорания итерации.

Доска задач

Доска задач выполняет двойную роль — дает команде удобный механизм организации работ и обеспечивает возможность увидеть, как много работы осталось. Очень важно, чтобы доска задач (или в некоторых случаях ее аналог) допускала значительную гибкость в организации работы команды. Члены agile-команды не берут на себя новую работу (или не получают новое задание) до тех пор, пока они не будут готовы взяться за нее. Это означает, что вплоть до последних одного-двух дней итерации обычно имеется множество нераспределенных задач. Доска делает эти задачи хорошо видимыми, так что у каждого есть возможность узнать, над чем предстоит работа и какую работу можно взять на себя. Пример доски задач приведен на рис. 20.1.

История	Для разработки	Тесты готовы	В работе	На проверку	Часы
<div>Как пользователь я могу...</div> <div>5</div>	<div>Кодирование...</div> <div>8</div> <div>Кодирование...</div> <div>5</div> <div>Тестирование...</div> <div>6</div>	✓	<div>Кодирование...</div> <div>SC 6</div> <div>Кодирование...</div> <div>DC 4</div>	<div>Кодирование...</div> <div>LC 4</div>	33
<div>Как пользователь я могу...</div> <div>2</div>	<div>Кодирование...</div> <div>8</div> <div>Кодирование...</div> <div>5</div>				13
<div>Как пользователь я могу...</div> <div>3</div>	<div>Кодирование...</div> <div>3</div> <div>Кодирование...</div> <div>6</div>	✓	<div>Кодирование...</div> <div>MC 4</div>		13

Рис. 20.1. Доска задач, используемая во время итерации

Доска задач нередко представляет собой большую магнитно-маркерную доску или, что лучше, пробковую доску. К доске прикрепляются липкой лентой или кнопкой карточки историй, а также карточки задач, которые составляются во время планирования итерации. Для каждой истории или функции, которая будет разрабатываться в процессе итерации, на доске задач отводится один ряд. На рис. 20.1 первый ряд содержит информацию о пятипунктовой истории. В первой колонке находится карточка истории. Поскольку на карточке истории указывается оценка истории в пунктах, любой взглянувший на доску задач может быстро определить количество пунктов для каждой истории, включенной в итерацию.

Во второй колонке находятся карточки всех задач, которые команда идентифицировала как необходимые для реализации истории или функции. На каждой из этих карточек обозначена оценка работы, оставшейся до завершения задачи.

Третья колонка показывает, готовы ли приемочные тесты для истории. Я активный сторонник разработки через тестирование (Beck, 2002) как на уровне кода, где рекомендую разработчикам писать модульные тесты до написания кода, так и на уровне функции, где рекомендую командам создавать общие приемочные тесты до начала кодирования. Если условия удовлетворенности для каждой истории идентифицируются в процессе

планирования итерации (как рекомендуется в главе 14 «Планирование итерации»), это не представляет трудности, поскольку условия удовлетворенности по своей сути — это общие приемочные тесты для пользовательской истории. Такой вид специфицирования на примерах очень удобен для программистов, которые могут ссылаться на конкретные примеры ожидаемой работы каждой функции и бизнес-правила.

Я предлагаю командам ставить большую галочку в колонке «Тесты готовы», когда они определяют общие тесты для истории. Помимо этого я рекомендую не перемещать много карточек в четвертую колонку «В работе» до тех пор, пока не определены тесты. Возможно, колонка «Тесты готовы» и не нужна, однако она служит наглядным напоминанием команде, которая пытается сделать правилом определение приемочных тестов до кодирования функции.

В колонку «В работе» помещают карточки, которые разработчики взяли в работу. Обычно разработчик берет карточку из колонки «Для разработки», ставит на ней свои инициалы и перемещает в колонку «В работе». Это происходит в тот день, когда разработчик завершает предыдущую работу и определяет, что ему делать дальше. Никто не должен брать более одной или двух карточек зараз. Это помогает поддерживать стабильный поток выполняемой работы и сокращает издержки, связанные с переключением с одной задачи на другую. Для напоминания об этом, когда команда устанавливает доску задач, я настаиваю на том, чтобы ширина колонки «В работе» составляла одну карточку. Колонка «Для разработки» обычно делается шире (шире, чем показано на рис. 20.1) по той причине, что карточки нередко приклеиваются по четыре в ряд.

Колонку «На проверку» можно делать, а можно и не делать, но я считаю ее полезной, особенно в тех случаях, когда работаешь с командой, осваивающей agile-подход. В идеале каждый тест продумывается, а каждая карточка задачи составляется во время планирования итерации. В этом случае при завершении задачи программирования («Кодирование пользовательского интерфейса») ее карточку удаляют с доски задач (или перемещают в последнюю колонку «Выполнено»). В этот момент кто-то может взять связанную с этой задачей карточку тестирования («Тестирование пользовательского интерфейса»). Вместе с тем бывает, что разработчик считает задачу выполненной, но хочет, чтобы кто-нибудь взглянул на нее свежим взглядом и проверил. В таких ситуациях и когда нет связанной задачи тестирования карточку задачи помещают в колонку «На проверку».

Разработчикам разрешается изменять оценку любой карточки задачи на доске в любой момент. Например, если я начинаю работать с карточкой и понимаю, что оценка «два часа» ошибочна, то иду к доске задач, зачеркиваю «два» и заменяю, скажем, на «шесть». Если, по моему мнению,

оценка еще больше, то я могу разделить эту задачу и заменить ее на две или три карточки задач, каждая со своей собственной оценкой. Последняя колонка доски задач — это просто сумма часов работы, остающихся в функции или истории. Я обычно суммирую часы для каждого ряда по утрам. Суммарные показатели используются для построения диаграммы выгорания итерации, которая является вторым инструментом отслеживания прогресса итерации.

Ошибки отслеживания на доске задач

Многие команды, начинающие переход к agile-подходу при разработке, сталкиваются с большим числом унаследованных ошибок. Дело не только в большом списке ошибок, которые необходимо устранять «при случае», но и в том, что ошибки продолжают появляться с высокой частотой. У команд, переходящих на agile-процесс, возникает вопрос, что с ними делать. Доска задач — удобный инструмент, помогающий приступить к устранению этой проблемы.

В качестве примера того, как доска задач может помочь, рассмотрим случай, когда владелец продукта включает задачу «Устранить 10 старых ошибок» в новую итерацию. Владелец продукта выбирает 10 ошибок, а разработчики составляют карточку задачи (с оценкой) для каждой из них. Эти карточки размещают в один ряд в колонке «Для разработки» доски задач. В процессе осуществления итерации разработчики занимаются этим десятком ошибок точно так же, как другими задачами. Теперь предположим, что пользователь находит новую ошибку в середине итерации. Если новой ошибке присваивается более высокий приоритет, чем у оставшихся в колонке «Для разработки», то владелец продукта может перенаправить эквивалентный объем работы на устранение новой ошибки.

Такой подход позволяет команде заниматься устранением унаследованных дефектов со скоростью, устанавливаемой владельцем продукта. Команда может выделить как 40 часов на устранение ошибок, так и все 100. Владелец продукта решает, сколько итераций следует посвятить устранению ошибок вместо разработки новых функций. Затем владелец продукта и команда совместно выбирают ошибки, подходящие по размеру для такого времени.

Диаграммы выгорания итерации

Построение диаграммы выгорания релиза — отличный способ понять, оторвался проект от плана или нет. В зависимости от длины итераций создание диаграммы выгорания может быть полезным и применительно к итерациям. Если вы работаете с однонедельными итерациями, то диаграмма

не нужна. К тому моменту, когда тренд на диаграмме выгорания станет видимым, однонедельная итерация уже закончится. Однако мой опыт показывает, что диаграммы выгорания очень полезны, когда длина итерации составляет две недели и более. Диаграмма выгорания итерации показывает оставшиеся часы по дням, как видно на рис. 20.2.

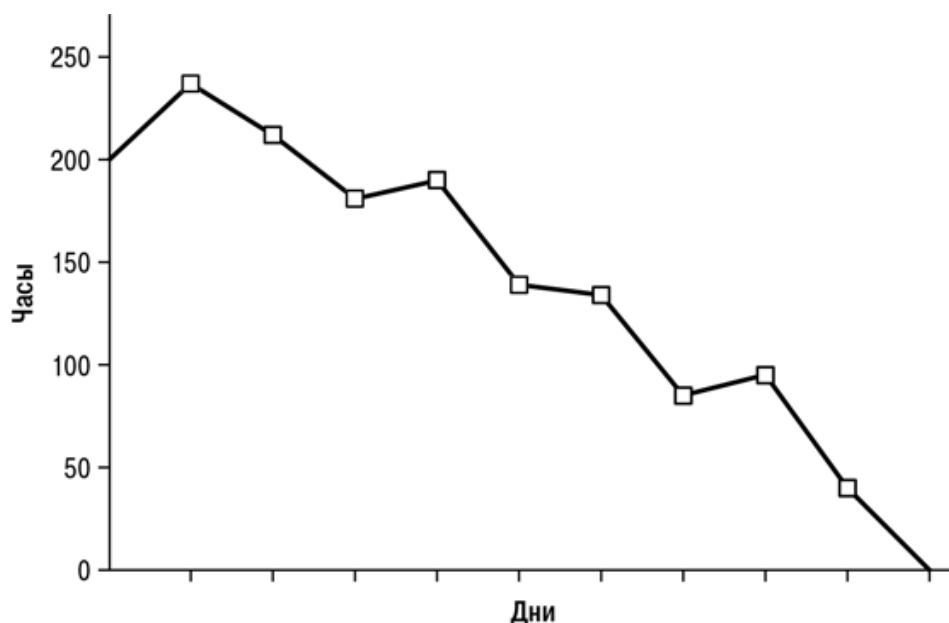


Рис. 20.2. Диаграмма выгорания итерации

Для построения диаграммы выгорания итерации просто суммируйте все часы на вашей доске задач один раз в день и наносите результат на график. Если у команды магнитно-маркерная доска задач, то я обычно строю диаграмму выгорания от руки на ее краю. Если доска задач пробковая, то я прикалываю к ней большой лист бумаги и строю диаграмму выгорания на нем.

Отслеживание затраченных сил и времени

В предыдущей главе была представлена аналогия проекта — парусная лодка, которая наглядно показывала, что пройденный путь не всегда легко измерить. Парусная лодка, которая вчера находилась в движении восемь часов, а затем стала на якорь, вовсе не обязательно находится на восемь часов ближе к пункту назначения. Ветер и течение могли отклонить лодку от того, что считалось ее курсом. Лодка с равным успехом может оказаться как ближе к пункту назначения, так и дальше от него. Когда такое происходит, самое разумное, что может сделать экипаж, это оценить свое положение относительно пункта назначения. Измерение пройденного

расстояния или затраченного времени не поможет, если нет уверенности в том, что продвижение идет в правильном направлении.

При осуществлении проекта намного полезнее знать, сколько осталось сделать, а не сколько сделано. Более того, отслеживание затраченных сил и времени и сравнение их с оценочными затратами может привести к «опасениям в отношении оценки» (Sanders, 1984). Когда оценщики боятся давать оценку, срабатывает знакомый инстинкт «бей или беги», и оценщики полагаются больше на него, а не на аналитическое мышление (Jørgensen, 2004).

Отслеживание затраченных сил и времени в целях повышения точности оценки — другое дело. В этой ситуации оно может быть полезным (Lederer and Prasad, 1998; Weinberg and Schulman, 1974). Вместе с тем руководитель проекта или тот, кто занимается таким отслеживанием, должен быть предельно осторожным и не допускать чрезмерного давления на оценщиков, поскольку оно ведет не к улучшению, а к ухудшению оценок.

Кроме того, необходимо помнить, что разброс — это неотъемлемая часть любой оценки. Сколько бы усилий ни вкладывалось в улучшение оценок, команда никогда не добьется идеальной оценки. Свидетельством этого могут служить хотя бы ваши утренние поездки на работу. Время в пути всегда будет варьировать независимо от того, на чем вы перемещаетесь, как далеко едете и где живете. Если вы добираетесь до работы на машине, каким бы ни было ваше водительское мастерство, оно все равно не поможет устранить разброс.

Индивидуальная скорость

Некоторые команды обращаются к индивидуальной скорости, т.е. к количеству пунктов или идеальных дней, реализуемых отдельным членом команды. Мой совет — перестаньте отслеживать индивидуальную скорость. Ее контроль приводит к поведению, которое мешает успешному осуществлению проекта. Допустим, вы объявляете о том, что будете измерять индивидуальную скорость и отслеживать ее от одной итерации к другой. Как, по-вашему, на это отреагируют члены команды? Если мне нужно будет выбирать между самостоятельным завершением истории и оказанием помощи кому-нибудь еще, то к чему, спрашивается, меня подтолкнет измерение индивидуальной скорости?

Для людей необходимо создавать стимулы, подталкивающие их к коллективной работе. Если производительность команды повышается от того, что я помогаю кому-то, то именно это я и должен делать. Значение имеет скорость команды, а не индивидуальная скорость. Индивидуальная

скорость не заслуживает даже мимолетного интереса.

Еще одним аргументом против измерения индивидуальной скорости является то, что ее просто невозможно рассчитать. Большинство пользовательских историй должны быть написаны так, чтобы они предполагали работу над ними нескольких человек, например дизайнера пользовательского интерфейса, программиста, администратора баз данных и тестировщика. Если большинство ваших историй реализуются одним человеком, вам следует пересмотреть подход к их составлению. Обычно это говорит о необходимости переписать их на более общем уровне так, чтобы в работе над ними могли участвовать несколько человек.

Резюме

Доска задач, которая нередко представляет собой магнитно-маркерную доску, пробковую доску или определенное место на стене, помогает команде организовать и визуализировать ее работу. Колонки на доске задач имеют определенные названия, и по мере выполнения работы члены команды перемещают карточки задач из одной колонки в другую.

Диаграмма выгорания итерации аналогична диаграмме выгорания релиза, но применяется для отслеживания работы только в текущей итерации. На ее вертикальной оси откладывают количество оставшихся часов, а на горизонтальной оси — дни итерации.

Командам не рекомендуется заниматься отслеживанием фактически затраченных на задачу часов для улучшения оценок. Риски и усилия, связанные с этим, обычно перевешивают получаемые выгоды.

Командам не следует подсчитывать или отслеживать индивидуальную скорость.

Вопросы для обсуждения

1. Будут ли в вашей организации отслеживание фактических затраченных на задачи усилий и сравнение их с оценками приносить выгоды, перевешивающие связанные с этим риски и затраты?
2. Если в вашей нынешней проектной команде нет предварительного распределения задач, что можно сделать для получения хотя бы части тех выгод, которые дает командам применение доски задач?

Глава 21

Информирование о плане

Чем сложнее наши средства коммуникации, тем меньше мы общаемся.

Джозеф Пристли

В этой главе мы рассмотрим конкретные способы информирования о планах. Важнее, однако, не то, о чем мы конкретно информируем, а то, как мы подходим к информированию об оценках и планах. Необходимо, чтобы любая коммуникация, и особенно коммуникация, связанная с оценками и планами, была частой, честной и двухсторонней.

Частое предоставление информации о планах важно из-за частоты обновления agile-плана. Например, даже если команда осуществляет скользящее опережающее планирование (как описано в главе 18 «Планирование проекта с участием нескольких команд»), ее план релиза может показывать только то, что будет разрабатываться в следующих нескольких итерациях. Пользовательские истории, которые будут разрабатываться в оставшейся части релиза, могут отражаться в его плане, но без определения очередности их выполнения за горизонтом опережающего плана и как широкие темы.

Мы не можем (да и не хотим) создавать план в первый день и оставлять его неизменным на протяжении трех–шести месяцев. Планы обновляются в течение всего проекта, и об этих обновлениях необходимо информировать заинтересованные стороны. Без информирования нет ценной обратной связи, которая может повысить желательность и успешность продукта, а также полезность плана. В течение короткой итерации членам команды важно видеть ежедневные изменения диаграммы выгорания с тем, чтобы вносить коррективы, необходимые для завершения всей запланированной работы. На протяжении более длительного срока релиза заинтересованным сторонам проекта и его участникам необходимо получать представление о прогрессе и изменениях плана релиза.

Честное информирование важно, если команда-разработчик и команда-клиент хотят доверять друг другу. Если разработчики узнают, например, что владелец продукта устанавливает для них необоснованные сроки, то они

перестают доверять ему. Аналогичным образом доверие исчезает, когда разработчики дают оценки, которые, насколько известно владельцу продукта, нереалистичны. Как-то я имел дело с командой, члены которой говорили, что руководство завышает для них объем работы, действуя по принципу «если сделают 80%, то уже хорошо». Руководство опиралось на теорию о том, что оно в любом случае получит не более 80% от запрашиваемого, а потому нужно запрашивать больше.

Без доверия трудно поддерживать честную коммуникацию, поэтому к потере доверия следует относиться очень серьезно. Если разработчик знает, что задача занимает намного больше времени, чем ожидается в текущий момент, то ему необходимо поделиться этим знанием с остальными членами команды, включая ее руководителя. Если такая коммуникация не поощряется, проблемы остаются скрытыми дольше.

Важно, чтобы коммуникация по вопросам оценки и планирования была двухсторонней, поскольку для получения наилучшего плана (с учетом текущего знания) и создания стоимости для организации необходимы диалог и дискуссия. Нам нужен итеративный подход и улучшение планов на основе обратной связи и новых знаний. Именно поэтому необходим именно диалог, а не односторонние презентации.

Наконец, возьмите в привычку убеждаться в том, что все получатели информации понимают ваше послание. Они должны не только услышать это послание, но и понять его. Если вы как руководитель проекта сообщаете о том, что проект отстает от календарного графика, сделайте это так, чтобы данная мысль дошла до каждого. Это одна из причин, по которым agile-команды предпочитают большие, наглядные графики в качестве инструмента коммуникации — нужно всего лишь несколько больших, наглядных графиков в рабочем помещении, тогда все члены проектной команды будут понимать, что к чему. Если же новость о том, что проект отстает от графика, «ясно» изложить на 32-й странице пухлого еженедельного отчета, то о ней, скорее всего, никто и знать не будет.

С учетом этих целей в оставшейся части настоящей главы излагаются конкретные правила и рекомендации по информированию об оценках и планах.

Информирование о плане

Когда спрашивают о сроке выполнения проекта, очень соблазнительно просуммировать количество поставляемых пунктов, разделить их на предполагаемую скорость и сказать что-нибудь вроде следующего: «Мы отгрузим продукт 14 июня, т.е. через 7,2 итерации от настоящего момента».

Это неправильно, поскольку создает впечатление, что знаний, на основе которых мы составляем план, достаточно для поддержки подобной оценки. Когда это возможно, указывайте в информации о целевой дате либо степень вашей уверенности в оценке, либо диапазон возможных дат, либо и то и другое. Например, вы можете сказать:

- «Я на 90% уверен в том, что мы завершим реализацию запланированной функциональности к 31 июля».
- «Исходя из наших предположений относительно размера проекта и производительности команды, на проект потребуется от трех до четырех месяцев».

В качестве примера информирования Рон Джеффрис (Jeffries, 2004) из www.XProgramming.com предлагает такой вариант:

В данный момент это похоже на 200-пунктовый проект. Исходя из нашей производительности в других проектах (или произвольного предположения), при участии N программистов в работе и вашей активной помощи на проект такого размера потребуется от четырех до шести месяцев. Вместе с тем мы будем поставлять вам программное обеспечение каждые две недели и делать все, чтобы реализованные истории удовлетворили вас. Хорошая новость заключается в том, что в случае неудовлетворенности вы можете остановиться. Еще лучше то, что в случае удовлетворенности, до того как будут реализованы все функции, вы также можете остановиться. Плохая новость заключается в том, что вам необходимо работать с нами, иначе мы не сможем узнать, что именно обеспечивает вашу удовлетворенность. Лучше всего то, что в ситуации, когда работающих функций достаточно для превращения программы в нечто полезное, вы можете попросить подготовить ее к выпуску, и мы сделаем это. В процессе нашего продвижения вперед станет очевидной быстрота прогресса и мы сможем уточнить свою оценку времени. В любом случае вы будете знать, что происходит, видеть конкретные результаты тестов, определенных вами, и получать всю информацию одновременно со мной.

Некоторые компании практикуют представление календарных графиков проектов в виде знакомой всем диаграммы Ганта, которая представлена на рис. 21.1. Диаграммы Ганта пользуются дурной репутацией, однако это связано с тем, что их часто применяют для предсказания, планирования и координирования задач. Несмотря на это, диаграммы Ганта могут быть очень полезным инструментом для отображения распределения функций по итерациям.



Рис. 21.1. Представление содержания итерации на диаграмме Гантта

Между диаграммой, представленной на рис. 21.1, и традиционной диаграммой Гантта есть несколько небольших, но принципиально важных отличий. Во-первых, детализация информации на рис. 21.1 ограничивается уровнем функции и разбивка пользовательских историй на задачи не производится. Таким образом, показывается *структура функций*, а не структура работ по проекту. Иначе говоря, диаграмма Гантта отображает функции, повышающие ценность продукта, а не задачи, из которых они состоят.

Во-вторых, каждая из показанных функций занимает полную итерацию, в которой она реализуется. Разработка функции вполне может быть завершена в середине итерации, но она все равно остается недоступной для организации до окончания данной итерации, именно это и отражает диаграмма Гантта.

В-третьих, на рис. 21.1 не показано распределение ресурсов. Поскольку за поставку всех функций отвечает команда в целом, нет смысла ставить имя Мэри напротив одной функции, а имя Вадим — напротив другой. Конечно, если диаграмма Гантта строится для отображения работы нескольких команд, то можно указать, что за такие-то функции (или чаще целые итерации) отвечает команда 1, команда 2 и т.д.

Информирование о прогрессе

Естественно, основным способом информирования о прогрессе являются диаграммы выгорания релиза, рассмотренные в главе 19 «Мониторинг

плана релиза». Прогресс на такой диаграмме — это функция количества оставшейся работы и скорости команды. Чтобы предсказать число оставшихся итераций, нужно взять количество оставшихся пунктов, разделить его на скорость команды и округлить результат до следующего большего целого числа. Иначе говоря, если остается 100 пунктов, а скорость команды равна 10, то остается 10 итераций.

Поскольку измерение скорости не отличается точностью, всегда следует ожидать колебания ее значения. Если команда определила, что ее скорость равна 10, не стоит сомневаться в том, что это средняя величина, и в следующих итерациях она может составлять как девять, так и 11. Не стоит удивляться, если на практике скорость в следующих итерациях будет варьировать от семи до 13. В таких случаях количество оставшихся итераций может колебаться от восьми до 15. При прогнозировании количества оставшихся итераций обычно лучше всего использовать диапазон вероятных скоростей.

Чтобы понять, как выбирается диапазон скоростей, рассмотрим рис. 21.2, где показана скорость команды в восьми прошлых итерациях. Скорость в последней итерации равна 19. Вместе с тем средняя скорость за восемь итераций составляет 17, а средняя скорость за три самые плохие итерации равна 14. Эти значения приводят к разным ожиданиям относительно срока завершения всех запланированных в текущий момент работ. Именно поэтому зачастую лучше использовать более одного значения скорости и представлять выводы в виде диапазона вероятных результатов.

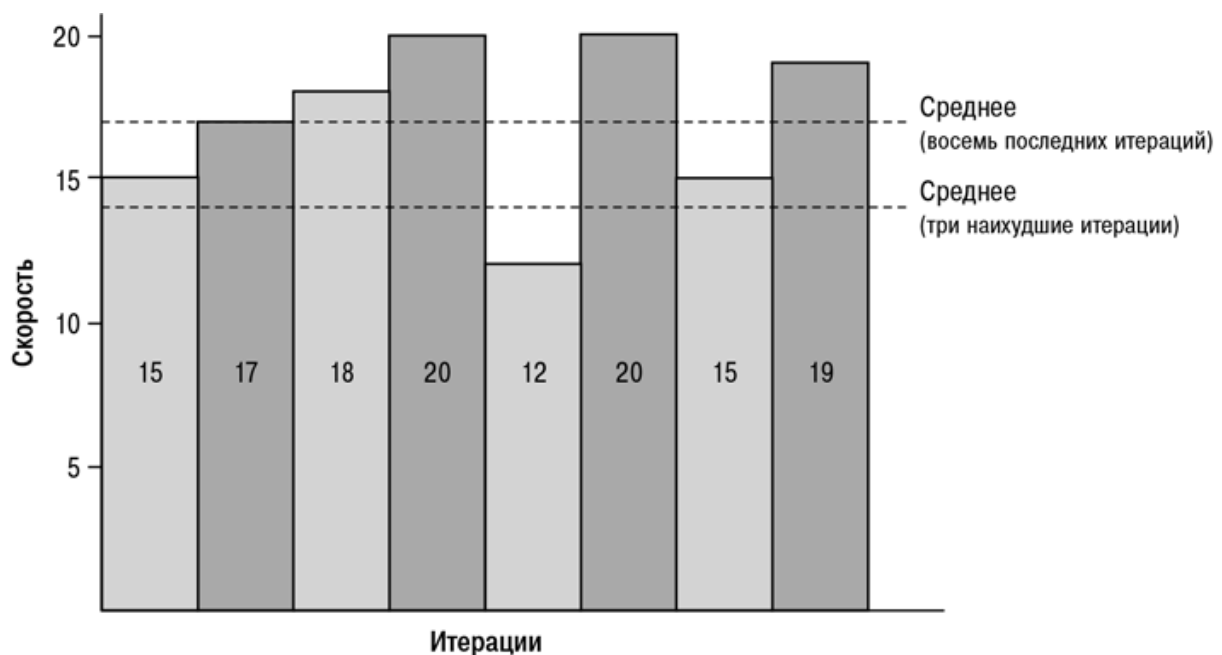


Рис. 21.2. Скорость, измеренная в прошлых восьми итерациях

При выборе диапазона значений скорости я обычно рассматриваю до восьми предыдущих итераций. Многие команды работают вместе и дольше, однако использование более восьми итераций, на мой взгляд, означает углубление в слишком древнюю историю. Если же у вас нет восьми итераций, используйте столько, сколько есть. В этих прошлых итерациях я выбираю три значения:

1. Скорость в последней итерации.
2. Среднюю скорость.
3. Среднюю скорость в трех итерациях с самой низкой скоростью.

Эти три значения представляют хорошую картину только что произошедшего — «долгосрочное» среднее и наихудший вариант того, что может произойти. На основе этих трех значений скорости я примерно прогнозирую, какой объем работы может быть выполнен к определенной дате. Например, если мы хотим выпустить продукт после пяти дополнительных итераций, а в последней итерации скорость команды составила 19, то можно рассчитывать на то, что команда выполнит еще 95 пунктов. Аналогичные вычисления можно проделать и с другими значениями скорости. В результате мы получим диапазон вероятных объемов работы, которую команда может выполнить. Итог я обычно представляю в табличной форме (табл. 21.1).

Таблица 21.1. Ожидаемое количество пунктов, определенное на основе скоростей из рис. 21.2

Описание	Скорость	Итерации	Суммарное количество пунктов
Наихудшие три	14	5	70
Последние восемь	17	5	85
Последняя итерация	19	5	95

Я пытаюсь определить тренды скорости, однако во многих проектах слишком мало итераций, чтобы тренды скорости проявились или были адекватными. Заметив что-то похожее на повышательный тренд скорости, я радуюсь, однако не полагаюсь на него. Я никогда не рисую линию тренда на рис. 21.2, например, и не говорю, что в следующей итерации скорость команды увеличится до 20. Аналогичным образом, если скорость, похоже, формирует нисходящий тренд, я учитываю это и стараюсь устранить причину падения, а не строю планы в расчете на падение скорости.

Расчет трех ожидаемых значений скорости (как показано в табл. 21.1) позволяет владельцу продукта и команде предсказывать объем работы, который может быть выполнен до плановой даты выпуска релиза. На

основании табл. 21.1, например, владелец продукта может быть уверен, что в последующих пяти итерациях будут добавлены как минимум 70 пунктов. Вместе с тем владелец продукта не должен рассчитывать на что-то большее, чем 85 пунктов.

Итоговый отчет в конце итерации

Может показаться, что agile-подход не предполагает подготовки каких-либо отчетов в конце итерации. Однако практически все руководители, с которыми я разговаривал, интересовались, составляю ли я его. Когда я отвечаю на этот вопрос утвердительно, меня почти всегда просят показать образец. Вы должны сами решить, стоит ли тратить время на такой отчет. Моя практика показывает, что на подобный итоговый документ уходит меньше получаса на итерацию. Для большинства проектов я рекомендую использовать двухнедельные итерации, а это означает, что на подготовку такого отчета тратится от силы 15 минут в неделю.

В последующих разделах представлен образец итогового отчета об итерации для проекта SwimStats. Обычно я включаю в отчет бóльшую часть рассмотренной ниже информации, однако вы можете по своему усмотрению удалять или добавлять разделы.

Контекст

Даты

Первый день итерации	1 сентября
Последний день итерации	14 сентября
Количество рабочих дней	9

Персонал

Указывается, кто из работников доступен во время итерации, а также ожидаемое количество дней, в течение которых они будут использоваться, и фактически отработанное количество дней.

Имя работника	Планируемое количество дней	Фактически отработанные дни
Карина	9	9
Вадим	9	7
Саша	8	9
Дмитрий	9	9
Всего	35	34

Рабочие дни

Поскольку выходные и праздники сокращают количество рабочих дней в итерации, в некоторых итерациях количество планируемых рабочих дней может варьировать. В нашем примере Вадим отработал на два дня меньше, чем планировалось. Возможно, он болел. В то же время Саша отработала на один день больше запланированного. Предположительно она собиралась взять выходной, но передумала.

Показатели

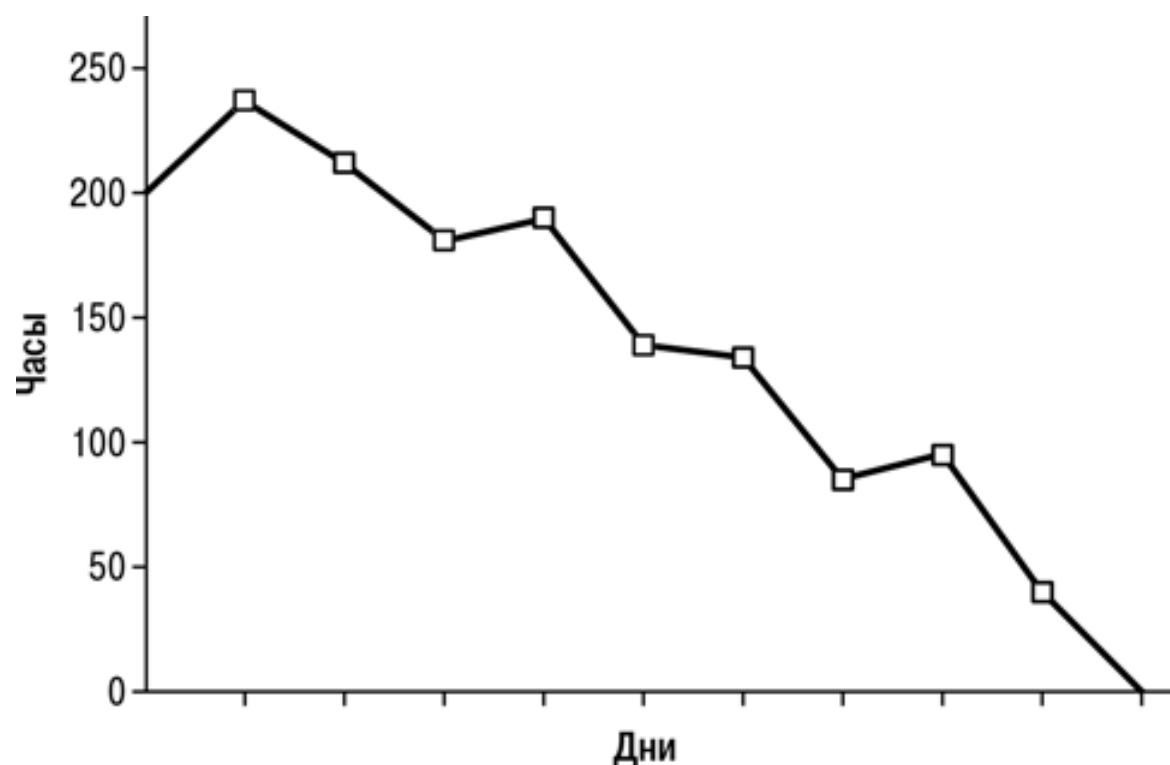
Результаты ночной сборки

Дата	Статус
Понедельник, 6 июня	Сборки не было
Вторник, 7 июня	812 модульных тестов пройдено 1431 FitNesse-тест пройден 104 теста пользовательского интерфейса пройдено
Среда, 8 июня	826 модульных тестов пройдено 1433 FitNesse-теста пройдено 104 теста пользовательского интерфейса пройдено
Четверг, 9 июня	1 модульный тест не пройден
Пятница, 10 июня	841 модульный тест пройден 1442 FitNesse-теста пройдено 105 тестов пользовательского интерфейса пройдено
...	

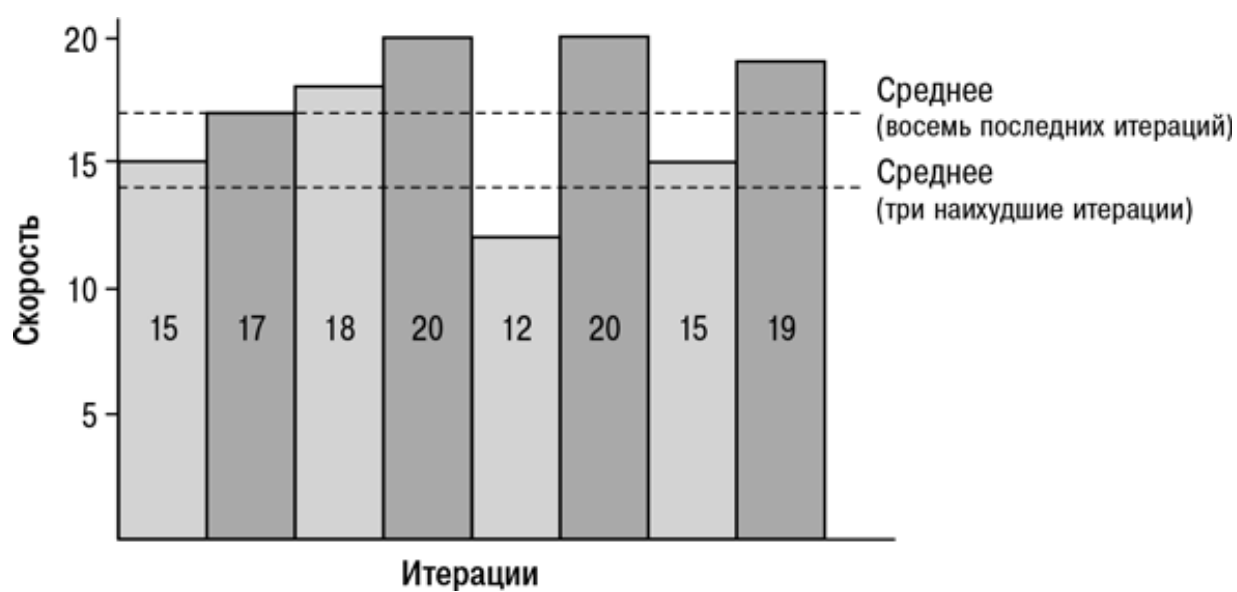
Цвет строк

Строки в этой таблице обычно делают цветными: зеленый — для успешной сборки, красный — для неудачной сборки. Обратите внимание на то, что в колонке «Статус» указывается число успешных тестов, только если все тесты успешно пройдены. Если хотя бы один тест не пройден, то все тесты попадают в разряд неудачных. В случае провала одного из тестов не имеет значения, сколько тестов пройдено. Если написать, что из 100 тестов пройдено 99, то возникает соблазн считать, что, раз 99% тестов пройдено, мы работаем хорошо. Во избежание этого я указываю в отчете число пройденных тестов, если они все пройдены, или, в противном случае, число непройденных тестов.

Выгорание итерации



Скорость



Контент и оценка

История	Результаты	Плановые пункты	Реализованные пункты
Как тренер я могу вводить имена и гендерно-возрастную информацию по всем пловцам моей команды	Завершена	8	8
Как тренер я могу определять расписание тренировок	Завершена; история больше, чем планировалось	8	8
Как пловец я могу видеть свое время во всех соревнованиях	Завершена	3	3
Как пловец я могу корректировать свою гендерно-возрастную информацию	Начата; не завершена	1	0
Всего		20	19

Анализ итерации

Анализ итерации проводился в 9:00 15 сентября. Высказаны следующие предложения:

Предложение	Ответственный
Марк отметил, что в журналах планирования тренировок, используемых большинством тренеров, с которыми он знаком, информация перечисляется в обратном порядке по сравнению с тем, что мы представляем на экране. Нам нужно, чтобы этим экранам дали оценку несколько тренеров	Карина
Гэри предложил добавить историю о графическом представлении результатов пловца в соревновании	Карина

Резюме

Нам нужно, чтобы информирование об оценках и планах было частым, честным и двухсторонним. Диаграмма Гантта может быть полезным инструментом информирования о плане. Вместе с тем ее не следует детализировать больше, чем до уровня функций, а функции должны представляться как находящиеся в работе на протяжении всей итерации.

Диаграммы выгорания — главное средство представления информации о прогрессе, однако их зачастую дополняют диаграммой скорости на итерацию для команды разработчиков. Полезно представлять скорость в виде диапазона, а не одного числа. С этой целью удобно использовать скорость в последней итерации, среднюю скорость в предыдущих восьми

итерациях и среднюю скорость в трех наименее результативных итерациях из предыдущих восьми. Эти три значения хорошо отражают то, что только что произошло, долгосрочное среднее значение и наихудший из возможных сценариев.

В некоторых проектах итоговые отчеты по итерациям могут быть полезными как для распространения текущей информации, так и в качестве документа, которым можно воспользоваться в будущем.

Вопросы для обсуждения

1. Предположим, что в проекте оставшийся объем работы оценивается в 150 пунктов. В последних 10 итерациях команда имела скорость 10, 12, 13, 5, 14, 7, 6, 12, 16 и 14. Вас спрашивают, когда будет завершен проект. Как вы ответите на этот вопрос?
2. Как установлен срок в вашем текущем проекте — как конкретная дата (скажем, 18 сентября) или как диапазон? Почему?
3. В каком диапазоне итераций или дат будет завершен ваш текущий проект?

Часть VI

Почему работает agile-подход к планированию

Если вам все еще непонятно, почему работает agile-подход к оценке и планированию, то мы постараемся объяснить это в настоящей части. Она содержит всего одну главу, посвященную исключительно этому вопросу. В этой главе мы быстро освежим в памяти, в чем заключается цель планирования, и рассмотрим наиболее важные причины, по которым agile-оценка и agile-планирование обеспечивают успешное достижение данной цели.

Глава завершается финальным набором правил применения agile-подхода к оценке и планированию в ваших проектах.

Глава 22

Почему работает agile-подход к планированию

Если вы хотите гарантию, купите тостер.

Клинт Иствуд в фильме «Новичок»

Дойдя до этой главы, мы получили достаточно знаний, чтобы понять, почему agile-подход к оценке и планированию обеспечивает успех. Прежде всего вспомните цель оценки и планирования. Планирование — это попытка оптимальным образом сбалансировать три аспекта проблемы разработки продукта: функции, ресурсы и сроки. Изменение одного из этих аспектов ведет к изменению других. При планировании мы изучаем весь спектр возможных соотношений этих аспектов в целях создания наилучшего продукта. Оценки и планы, которые мы создаем, должны быть достаточными, чтобы служить основой для принятия решений организацией. В настоящей главе мы рассмотрим, как agile-подход к оценке и планированию, описываемый в этой книге, помогает достичь названных целей.

Частое изменение плана

Одним из путей обеспечения эффективного исследования сферы разработки нового продукта, характерных для agile-подхода к оценке и планированию, является частое изменение плана. В начале каждой новой итерации для нее создается план. План релиза обновляют либо после каждой итерации, либо (в худшем случае) после каждых нескольких итераций. Признание невозможности создания идеального плана значительно ослабляет стремление к достижению такой цели (Githens, 1998). Уверенность в том, что план можно пересмотреть в начале следующей итерации, смещает внимание команды с идеи создания идеального (и недостижимого) плана на идею создания плана, полезного в текущий момент.

Чтобы план оказался полезным, он должен быть точным, однако мы

допускаем, что первоначальные планы неточны. Одна из причин изменения плана — постепенное уменьшение неточности. Иными словами, первоначальный план может говорить, что в третьем квартале в проекте будет выполнена работа объемом 300–400 пунктов. Этот план впоследствии может оказаться более-менее точным (в августе выполняется работа объемом 305 пунктов), но он все равно неточен. В начале проекта такой план, скорее всего, служит основой для принятия решений. Позднее, однако, чтобы оставаться полезным, план уточняется, и мы можем сказать, что в сентябре в проекте будет реализовано 380–400 пунктов.

При agile-подходе к оценке и планированию мы признаем, что наше знание всегда неполно, а следовательно, планы необходимо пересматривать по мере обучения. По мере того, как команда больше узнает о продукте в процессе его разработки, в план релиза могут добавляться новые функции. По мере того, как команда узнаёт больше об используемых технологиях или об их совместной работе, корректируются ожидания в отношении скорости ее прогресса. Чтобы план оставался полезным, в него необходимо встраивать новые знания.

Оценки размера и сроков разделяются

Обычная ошибка планирования (как у традиционных, так и у многих agile-команд) — смешивание оценок размера и срока. Понятно, что размер работы и время, необходимое для ее выполнения, взаимосвязаны, однако сроки зависят от множества дополнительных факторов. На проект одного размера программистам с разной квалификацией и опытом потребуется разное время. Аналогичным образом на сроки влияет размер команды, занимающейся проектом. У команды из четырех человек на работу может уйти шесть месяцев (24 человеко-месяца). Команда из восьми человек может сократить срок до четырех календарных месяцев, но затратить на работу 32 человеко-месяца, несмотря на то что размер проекта один и тот же.

Чтобы понять разницу между оценками размера и срока, представьте, что я показываю вам книгу и спрашиваю, за какое время вы сможете прочесть ее. Из названия вы можете понять, что это роман, но я не даю вам раскрыть книгу и посмотреть, сколько в ней страниц, какие в ней поля и насколько мелкий шрифт. Чтобы ответить на мой вопрос, вы сначала оцениваете объем — количество страниц. Допустим, их 600. Затем вы прикидываете свою скорость чтения и решаете, что она составляет одну страницу в минуту. В результате вы говорите мне, что прочтете книгу за 600 минут, или за 10 часов. Чтобы назвать срок (10 часов), вы сначала

оцениваете объем работы (600 страниц).

Agile-подход к планированию приносит успех потому, что в нем оценки размера и сроков разделены. Мы начинаем с оценки в пунктах размера пользовательских историй в проекте. Поскольку пункты абстрактны и относительны, они представляют чистую оценку размера. Затем оценивается темп прогресса, который мы называем скоростью. Наконец, наши оценки размера и скорости объединяются и дают оценку срока. Процесс оценки и планирования только выигрывает от этого ясного и полного разделения оценок размера и срока.

Планы составляются на разных уровнях

Поскольку agile-планы охватывают три разных уровня — релиз, итерацию и текущий день, каждый из них может иметь свой уровень точности. Наличие планов с разными временными горизонтами и разными уровнями точности дает два преимущества. Во-первых, это показывает, что разные планы создаются по разным причинам. Дневной план, обязательство, по выполнению которого каждый участник принимает на ежедневном совещании команды, сравнительно точен: исполнители выражают готовность выполнить или как минимум продвинуться в выполнении конкретных задач в течение дня. План итерации менее точен — в нем перечисляются пользовательские истории, которые подлежат разработке в течение итерации, и задачи, которые считаются необходимыми для этого. Каждая пользовательская история определена неидеально, поэтому существует некоторая неопределенность в отношении того, что понимать под реализацией той или иной истории за итерацию. Наконец, план релиза — это наименее точный план, содержащий только приоритизированный перечень желательных пользовательских историй и одну или несколько оценок объема желательных функций, которые, вероятнее всего, необходимо поставить к желательной дате релиза.

Во-вторых, планирование на разных уровнях помогает команде смотреть на проект с разных точек зрения. План итерации необходим для обеспечения слаженной работы кроссфункциональной команды на протяжении короткой итерации. План релиза дает более широкое представление о проекте и не позволяет команде не заметить леса релиза за деревьями итераций. Команда, которая работает над итерацией за итерацией, не имея представления о более отдаленной цели, рискует замкнуться на краткосрочных целях и упустить реально выгодную долгосрочную цель. Краткосрочные цели могут противоречить желаемому долгосрочному результату.

Планы ориентируются на функции, а не на задачи

Традиционный план в форме диаграммы Ганта, диаграммы PERT или разбивки работ сфокусирован на задачах, выполнение которых необходимо для создания продукта. Agile-план концентрируется на функциях, которые должны присутствовать в продукте. Это принципиальное отличие, которое заставляет команду думать о продукте на правильном уровне — на уровне функций. Когда функции разрабатываются итеративно, уменьшается потребность в предварительном обдумывании необходимых конкретных задач. В процессе осмысления работы для очередной итерации команда обдумывает или открывает все задачи по мере их необходимости. Еще важнее то, что команда думает о функциях, которые подлежат разработке. Мой коллега Джим Хайсмит (Highsmith, 2004b) утверждает, что «agile-планирование “лучше” других потому, что оно ориентируется на функции (истории и т.п.), а не на задачи. Довольно легко спланировать целый проект, используя стандартные задачи, без реального понимания сущности создаваемого продукта. Когда планирование осуществляется на основе функций, команда намного лучше понимает продукт». На уровне задач планы многих проектов выглядят одинаково. Каждый agile-план конкретен по отношению к создаваемому продукту.

Небольшие истории поддерживают постоянство потока работы

Из теории массового обслуживания (Porpendieck and Porpendieck, 2003; Reinertsen, 1997) мы знаем о важности фокусирования на *времени цикла* — количестве времени, необходимого для чего-то от начала процесса до его завершения. В софтверном проекте время цикла — это время от начала работы команды над функцией до поставки этой функции пользователям. Чем короче время цикла, тем лучше.

Ключевым фактором для времени цикла является разброс времени, необходимого для разработки новой функции. Лучшим способом уменьшения разброса является выполнение небольших и примерно одинаковых по размеру блоков работы. Процесс оценки и планирования, описанный в книге, ориентирован именно на это, если вспомнить рекомендацию оценивать краткосрочную работу в пределах одного порядка. Крупные пользовательские истории могут существовать в конце списка приоритизированных требований проекта, однако по мере

приближения к началу списка (когда их включают в предстоящую итерацию) их необходимо разбивать на более мелкие части.

Незавершенная работа ликвидируется в каждой итерации

Большой объем незавершенной работы замедляет прогресс команды. В софтверном проекте незавершенная работа связана с какой-либо функцией, разработку которой команда начала, но еще не закончила. Чем больше объем незавершенной работы, тем больше времени требуется на реализацию новой функции, поскольку все новое должно начинаться после завершения уже начатой работы. (Бывают случаи, когда реализацию новой функции необходимо ускорить, перепрыгнув через незавершенную работу, однако это лишь усугубляет проблему для следующей функции, реализацию которой невозможно ускорить.) Поэтому незавершенная работа приводит к увеличению времени цикла — в предыдущем разделе мы уже говорили о важности поддержания короткого цикла.

Одна из причин, по которым agile-планирование достигает цели, заключается в том, что вся незавершенная работа ликвидируется в конце каждой итерации. Поскольку работа не переносится автоматически вперед из одной итерации в другую, каждую итерацию планируют с чистого листа. Это означает, что работа над функцией, которая не завершена полностью в одной итерации, не обязательно продолжается в следующей. Зачастую продолжается, но не всегда. В результате мы получаем эффект полного отсутствия незавершенной работы в начале каждой итерации.

Поскольку незавершенная работа ликвидируется в начале каждой итерации, командам легче добиться эффективной работы в коротких итерациях. Это означает более быстрое получение обратной связи от пользователей и, как следствие, более быстрое обучение проектной команды наряду с более своевременным контролем и уменьшением риска.

Отслеживание прогресса осуществляется на уровне команды

В традиционном подходе к оценке и планированию прогресс измеряется и вознаграждается на уровне отдельных членов команды. Это ведет к возникновению целого ряда проблем. Например, если досрочное выполнение задачи приводит к обвинению программиста в раздувании

оценки этой задачи, то программист перестает завершать что-либо досрочно. Вместо досрочного завершения он не будет представлять задачу как завершенную до истечения срока.

Agile-подход к оценке и планированию успешно устраняет эту проблему через отслеживание прогресса только на уровне команды. Это одна из причин, по которым в главу 14 «Планирование итерации» включена рекомендация для членов команды воздерживаться от принятия обязательств по выполнению конкретных задач во время планирования итерации. Аналогичным образом не должны составляться индивидуальные диаграммы выгорания — составляется только общекомандная диаграмма выгорания.

Неопределенность признается и учитывается при планировании

Многие традиционные директивные планы ошибочно исходят из того, что функции можно зафиксировать в самом начале проекта. После этого появляются планы, не допускающие изменений или ставящие на их пути барьер в виде обременительного процесса контроля. Это приводит к реализации проектов с функциями, которые не нужны пользователям. Когда планы составляются в начале проекта и не обновляются по мере приобретения новых знаний, теряется возможность синхронизировать их с реальностью.

При agile-подходе к оценке и планированию команды признают неопределенность и целей, и средств. Неопределенность целей (неопределенность, связанная с создаваемым в конечном итоге продуктом) уменьшается по мере представления дополнений продукта потенциальным пользователям и другим заинтересованным сторонам в конце каждой итерации. Обратная связь используется для тонкой корректировки будущих планов. Неопределенность средств (неопределенность, связанная с тем, как будет создаваться продукт) уменьшается по мере того, как команда больше узнает об используемых технологиях и о себе. Быстрое обнаружение того, что определенный компонент стороннего производителя не удовлетворяет требованиям по производительности, например, может заставить команду искать альтернативы. Время на поиск и переключение на новый компонент необходимо учитывать в новых планах после идентификации потребности.

Правила применения agile-подхода к оценке и

планированию

С учетом сказанного выше я составил список из 12 правил успешного применения agile-подхода к оценке и планированию.

1. **В осуществлении проекта должна участвовать вся команда.** За конкретные виды деятельности может отвечать один или несколько человек. Например, определение требований к приоритизации является функцией главным образом владельца продукта. Вместе с тем для создания наиболее ценного продукта при реализации проекта необходимо участие всей команды. Это следует, например, из того, что оценка лучше всего удастся команде в целом, хотя работать с оцениваемой историей или задачей будет лишь один или два члена команды. Чем больше ответственности берет на себя вся команда, тем более успешно она действует.
2. **Планируйте на разных уровнях.** Ошибочно думать, что план релиза делает план итерации ненужным, или наоборот. План релиза, план итерации и дневной план имеют разные временные горизонты и разные уровни точности. Каждый из них служит своей цели.
3. **Разделяйте оценки размера и сроков, используя разные единицы.** Проще всего разграничить оценку размера и оценку срока с помощью использования разных единиц измерения, которые нельзя спутать. Оценка размера в пунктах и превращение размера в сроки с помощью скорости — отличный способ добиться этого.
4. **Выражайте неопределенность при представлении либо функциональности, либо даты.** Ни один план не является полностью определенным. Обязательно включайте выражение неопределенности в составляемые планы релизов. Если объем новой функциональности зафиксирован, представляйте неопределенность в виде диапазона дат («Мы завершим работу в третьем квартале» или «Мы завершим работу через 7–10 итераций»). Если зафиксирована дата, представляйте неопределенность через набор поставляемых функций («Мы завершим работу 31 декабря, и продукт будет содержать по меньшей мере вот эти функции, но, возможно, не более, чем те новые функции»). Используйте более крупные единицы (итерации, месяцы, кварталы, например) по мере роста неопределенности.
5. **Часто пересматривайте планы.** Перед началом каждой итерации не упускайте возможность оценить релевантность текущего плана релиза. Если план релиза строится на основе устаревшей информации или на допущениях, которые стали некорректными, обновите его. Пользуйтесь любыми возможностями пересмотреть

планы с тем, чтобы проект всегда был нацелен на создание наибольшей стоимости для организации.

6. **Отслеживайте прогресс и информируйте о нем.** Многие люди, имеющие отношение к проекту, крайне заинтересованы в получении информации о прогрессе. Обеспечьте их регулярное информирование путем публикации простых и понятных индикаторов прогресса команды. Для этой цели очень хорошо подходят диаграммы выгорания и прочие наглядные индикаторы прогресса в проекте.
7. **Признавайте важность обучения.** Поскольку проект во многом связан с генерированием новых знаний о добавлении возможностей в продукт, планы необходимо обновлять с учетом этих новых знаний. По мере того как мы узнаем больше о потребностях наших клиентов, мы добавляем в проект новые функции. По мере приобретения знаний об используемых технологиях и о своей работе в команде мы корректируем ожидания в отношении темпов прогресса и желаемого подхода.
8. **Планируйте функции подходящего размера.** Функции, которые будут добавлены в ближайшем будущем (в течение следующих нескольких итераций), необходимо разбивать на относительно небольшие пользовательские истории — обычно такие, реализация которых занимает один-два дня, в любом случае не более 10 дней. Мы лучше всего оцениваем работу, имеющую размеры одного порядка. Работа с пользовательскими историями в таком диапазоне размеров обеспечивает наилучшее сочетание затрат и точности. Подобный подход также приводит к получению сравнительно небольших историй, которые большинство команд могут реализовать за одну итерацию. Конечно, работа с небольшими пользовательскими историями может оказаться довольно сложной в условиях длительного проекта. Во избежание этого, если вы создаете план релиза, рассчитанный более чем на два-три месяца, либо напишите несколько крупных историй (которые называют *эпопеями*), либо оценивайте более отдаленную работу на уровне темы, чтобы не пришлось разбивать крупные истории на части задолго до того, как это потребуется.
9. **Приоритизируйте функции.** Работайте с функциями в таком порядке, который оптимизирует суммарную стоимость проекта. В дополнение к стоимости и себестоимости функций учитывайте при приоритизации обучение, происходящее в процессе работы, и снижение риска в ходе разработки функции. Если разработка конкретной функции на раннем этапе позволяет команде значительно углубить знания о продукте или об усилиях,

необходимых для его создания, к ее реализации следует приступить как можно раньше.

10. **Стройте оценки и планы на фактах.** Когда возможно, стройте оценки и планы на реальной основе. Конечно, бывает, что в некоторых организациях приходится оценивать показатели вроде скорости, опираясь на очень узкую базу. В главе 16 «Оценка скорости» на этот случай представлены подходящие методики. Вместе с тем, когда возможно, оценки и планы должны строиться на реальных, наблюдаемых величинах. Это относится также и к оценке того, сколько функций реализовано. Несложно увидеть, что функция выполнена на 0% (в момент, когда с нею только начинают работать), и довольно легко определить, когда она выполнена на 100% (пройдены все тесты для всех условий удовлетворенности владельца продукта). Однако в промежутке между этими состояниями очень трудно сказать, на сколько реализована функция — на 50% или на 60%. Как результат, придерживайтесь того, что вам известно: 0% и 100%.
11. **Оставляйте небольшой резерв.** Особенно при планировании итерации не пытайтесь задействовать 100% времени каждого члена команды. Точно так же, как на шоссе возникает пробка при его загрузке на 100%, команда разработчиков начинает работать медленнее, когда время каждого члена команды полностью заполнено.
12. **Координируйте работу команд с помощью опережающего планирования.** В проекте с участием нескольких команд координируйте их работу, используя скользящее опережающее планирование. Заглядывая вперед и распределяя конкретные функции между конкретными итерациями, можно строить планы с учетом взаимозависимостей команд.

Резюме

Цель agile-планирования заключается в итеративном поиске оптимального ответа на общий вопрос разработки продукта — какие функции какими ресурсами и за какое время можно реализовать. Agile-подход к оценке и планированию позволяет успешно дать ответ на этот вопрос по следующим причинам: планы составляются на разных уровнях и часто пересматриваются, они строятся на основе функций, а не задач; сначала оценивается размер, а потом на основе оценки размера определяется срок; используются небольшие истории, обеспечивающие постоянный поток

работы, а незавершенная работа ликвидируется в конце каждой итерации; прогресс измеряется на уровне команды, а не индивидуального исполнителя; неопределенность признается и учитывается при планировании.

Вопросы для обсуждения

1. Есть ли другие причины, которыми, на ваш взгляд, объясняется успех agile-подхода к оценке и планированию?
2. Какие из 12 правил применимы к вашему текущему процессу оценки и планирования? Будет ли полезно следовать другим правилам?

Часть VII

Анализ конкретного примера

Все, что нужно, уже сказано, однако в единственной главе настоящей части я повторю это снова, но уже иначе.

В этой главе описывается вымышленная ситуация, однако в ней обобщаются многие ключевые моменты книги. В данной главе представлена выдуманная компания Bomb Shelter Studios, которая осуществляет свой первый agile-проект. В процессе повествования вы познакомитесь со следующими персонажами:

- Аллан — программист;
- Делани — аналитик;
- Карлос — agile-тренер;
- Лора — финансовый директор;
- Прасад — тестировщик;
- Роуз — художник;
- Саша — программист;
- Фил — генеральный директор;
- Фрэнк — менеджер по продукту.

Глава 23

Конкретный пример: Bomb Shelter Studios

Перелет занял много времени, но конференция оказалась полезной. Путь назад с Восточного побережья всегда был испытанием, но на этот раз Фрэнк летел первым классом, обменяв часть накопленных бонусных миль на повышенный комфорт. Фрэнк устроился в кресле и стал размышлять о событиях прошедшей недели.

Как менеджер по продукту в Bomb Shelter Studios, он знал, что их последняя игровая программа Deep Black & White демонстрировала хорошие результаты. Она позволяла играть в игру под названием го, которая была чрезвычайно популярна в Японии, Китае и Корее, но вызывала лишь умеренный интерес в Европе, Северной Америке и остальном мире. Программисты из его команды использовали решения на основе искусственного интеллекта, которые позволили Deep Black & White дойти в игре до уровня сложности, известного как пятый дан. Это, конечно, далеко от девятого дана — уровня лучших профессиональных игроков, однако значительно лучше, чем у конкурентов компании Bomb Shelter.

Фрэнк был полон энтузиазма в отношении выпуска и продажи Deep Black & White в Азии через одного издателя, с которым удалось договориться о дистрибуции во время конференции. Доход от продаж на этих рынках был бы очень кстати для Bomb Shelter. Фрэнк знал: дополнительные шесть месяцев, которые потребовались для завершения Deep Black & White, чуть было не привели к разорению небольшую частную компанию по разработке игр, соучредителем которой он являлся.

Из малоперспективной начинающей компании за три года Bomb Shelter Studios превратилась в признанного разработчика интеллектуальных и стратегических игр. В дополнение к только что законченной Deep Black & White у Bomb Shelter были программы для игры в шахматы, триктрак, реверси, бридж, шашки, манкалу и др. После создания игры права на ее дистрибуцию продавались тому или иному издателю, который занимался производством и распространением, позволяя Bomb Shelter полностью сконцентрироваться на разработке новых игр.

Пока Фрэнк был на конференции, его аналитик и небольшая команда в

Санта-Барбаре обдумывали новую игру Havannah, разработку которой они были практически готовы начать. Из-за проблем с Deer Black & White — не только из-за шестимесячной задержки, но и из-за слишком большого количества ошибок и обнаруженного в последний момент неудобства использования — им нужно было срочно найти другой подход к планированию и реализации проектов. Саша, ведущий разработчик архитектуры в компании, проработала ряд выдвинутых командой идей. Она предложила в следующем проекте использовать то, что называют «agile-процесс». Фрэнк не совсем понимал, что это означает, однако у него не было сомнений в том, что им нужно подходить к делу по-другому. Шестимесячное опоздание с выпуском следующей игры было бы концом. Все члены команды горели идеей попробовать agile-процесс и знали, что поставлено на кон.

День 1 — утро понедельника

— Всем привет, — сказал Фрэнк, входя в конференц-зал. До девяти оставалась еще целая минута, однако почти вся команда была в сборе и ожидала его появления. Это был хороший знак. Несмотря на то, что команда измоталась в процессе последнего рывка с Deer Black & White, она выглядела так, словно была готова взяться за разработку Havannah.

— Привет, Фрэнк. Я видел твоё электронное письмо о Deer Black & White. Сделка по дистрибуции — отличная новость, — сказал Аллан, программист C++, который разрабатывал игровой движок на искусственном интеллекте, сделавший Deer Black & White таким сильным игроком.

— Возьми пончик, Фрэнк, — предложила Саша и подвинула к нему почти пустую коробку.

— Спасибо, — ответил Фрэнк и запустил руку в коробку.

— Фрэнк, это Карлос, — сказала Саша. — Он опытный agile-тренер. Мы пригласили его, чтобы он помог нам освоить этот новый agile-подход к работе.

Фрэнк и Карлос обменялись рукопожатиями и приветствиями.

— Похоже, все в сборе, за исключением Роуз, — сказал Фрэнк. — Давайте начнем. Мы введем ее в курс дела позже. Вряд ли на этом совещании разговор пойдет о художественном оформлении.

— Мы не можем начать, Фрэнк, — сказал Карлос. — Это важный момент. Нужно, чтобы присутствовала *вся команда*. Значительная часть выгоды, которую мы хотим получить в результате применения agile-процесса, зависит от участия всех членов команды. Роуз, возможно, мало

что скажет об искусственном интеллекте в игровом движке, но нам все равно нужно знать ее мнение, если мы хотим создать лучшую игру.

— Она всегда появляется по понедельникам в пять минут десятого. Ей приходится отвозить Брук в школу по понедельникам, средам и пятницам. Она вот-вот придет, — заметил Прасад. И точно, стоило ему сказать это, как отворилась дверь и в конференц-зал вошла Роуз.

— Извините, пробки, — сказала Роуз, быстро усаживаясь в кресло.

— Итак, Делани, ты занималась исследованием продукта для проекта Havannah, — обратился Фрэнк к аналитику. — Прошло довольно много времени с того момента, как я задумал эту игру. Прошу прощения, но напомни мне, пожалуйста, как в нее играют.

— Конечно, Фрэнк. Прежде всего игровое поле выглядит вот так, — сказала Делани, вытащила деревянную доску из сумки и положила ее на стол (рис. 23.1). — В игре участвуют два игрока, которые по очереди ставят фишки на поле. У каждого игрока фишки своего цвета. После размещения фишки на поле двигать ее больше нельзя.

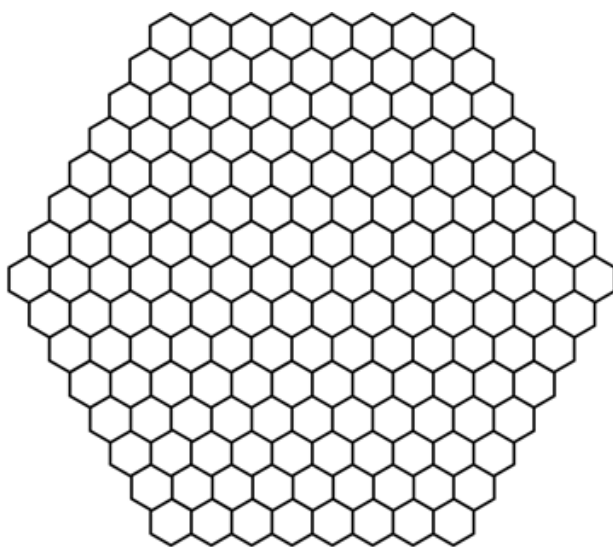


Рис. 23.1. Пустое поле игры Havannah

— Правильно, но, в отличие от го, фишки не захватывают. Цель игры Havannah — построить кольцо, мост или вилку. Кто первым это сделает, тот и выигрывает.

— А что такое кольца, мосты и вилки? — спросил Фрэнк, когда Делани взяла пригоршню фишек и начала ставить их на доску.

— Кольцо — самая простая фигура. Оно выглядит так, — ответила Делани, расставляя фишки, как показано на рис. 23.2. — Кольцо — это ряд стоящих на соседних клетках фишек, которые окружают одну или несколько клеток.

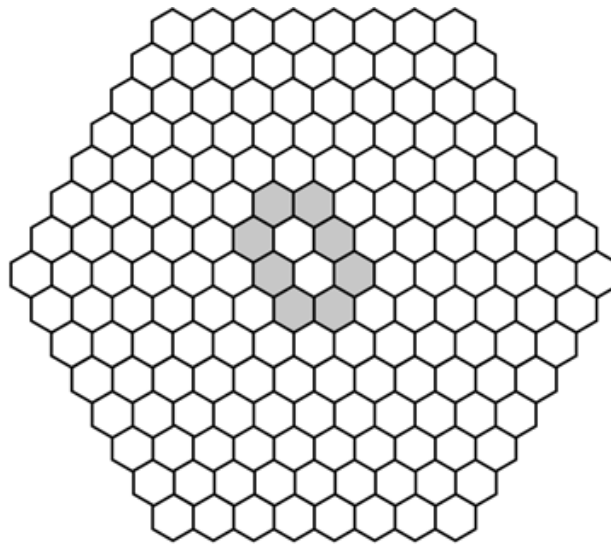


Рис. 23.2. Кольцо

— Ну, это выглядит довольно просто. Что усложняет игру? — спросил Аллан. Он уже начал думать о том, как создать движок на основе искусственного интеллекта, который будет выбирать ходы.

— Не забывайте, что кольцо — только один из путей к победе. Выиграть можно также, построив вилку или мост, — продолжила Делани, расставляя фишки, как показано на рис. 23.3. — Мост соединяет любые два угла. Он может представлять собой прямую линию, идущую по краю от одного угла к другому. Чаще, однако, мост выглядит так. — Она показала на мост в правой части доски на рис. 23.3.

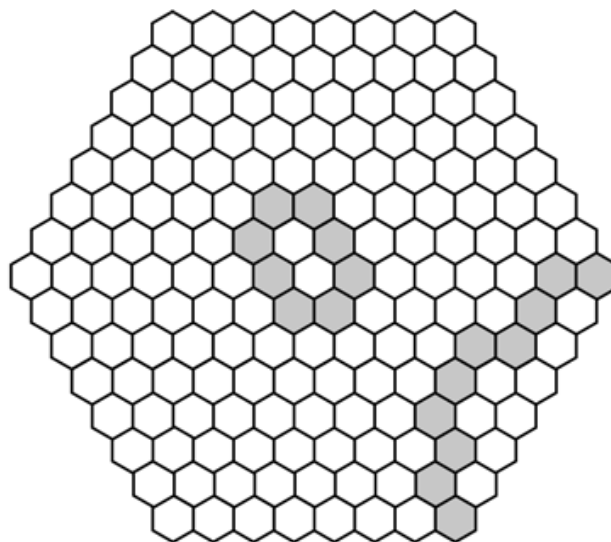


Рис. 23.3. Кольцо и мост

— Должен ли игрок говорить до начала игры, какую фигуру он попытается построить? — поинтересовался Аллан.

— Нет, это часть интриги и часть задачи. Ты можешь начать со строительства моста, понять, что он не получается, и попытаться построить

вилку.

— А что такое вилка?

— Вилка выглядит вот так, Фрэнк, — сказала Делани, добавив несколько фишек на доску, как показано на рис. 23.4. — Вилка соединяет три обреза доски, а не углы. Углы — это не обрезы, поэтому их нельзя использовать для создания вилки, они подходят только для моста.

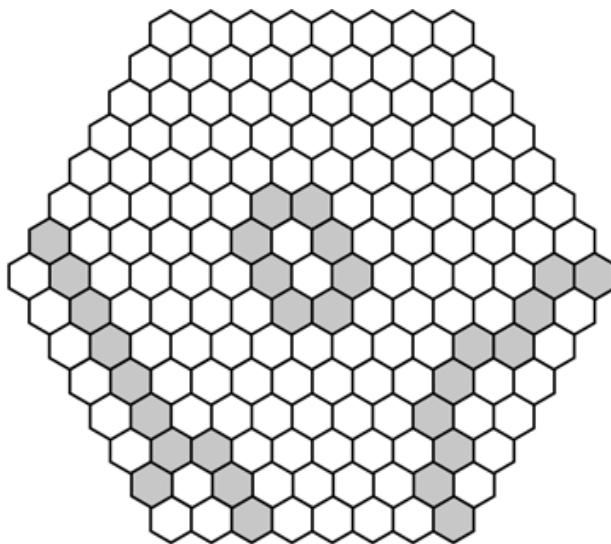


Рис. 23.4. Победные фигуры в игре Navannah: вилка, кольцо и мост

— Создание генератора ходов будет проблемой. При таком большом количестве возможных вариантов достижения победы и таком количестве клеток на доске число комбинаций огромно.

— Именно так, Аллан, — сказала Делани. — Многие считают, что эта игра сложнее, чем шахматы, поскольку в ней больше вариантов и невозможно использовать огромную базу эндшпилей. В шахматах, когда остается всего несколько фигур, можно использовать базу лучших завершений партии и не нужно полагаться только на генератор ходов. В игре Navannah слишком много фишек и слишком много позиций.

— Но ведь тебе неинтересно возиться с какой-нибудь простенькой игровой программой, правда же, Аллан? — поддела его Саша.

По виду Аллана было похоже, что в этот раз он не прочь заняться чем-нибудь попроще.

— Не волнуйся. Из-за того, что сейчас в эту игру мало кто играет, нам не понадобится такой же мощный движок, как в Deep Black & White, — поспешила успокоить его Делани. — В конечном итоге мы доберемся и до такого же уровня, но в версии 1.0 этого пока не нужно. Вполне подойдет движок, который берет верх над людьми в большинстве случаев.

— Окей, Делани. Есть ли еще какие-то правила, о которых нам нужно знать? — спросил Фрэнк.

— Нет, это все. Правила простые, но эта игра реально заставляет поломать голову. Именно поэтому мы считаем, что она понравится клиентам, которые уже покупали наши игры.

— Так ты уже оформила документ с требованиями?

— Пока нет, Фрэнк. Из того, что мы узнали от Карлоса об agile-подходе к разработке программного обеспечения, следует, что команда должна формулировать требования коллективно.

— Совершенно верно, — добавил Карлос. — Мы собираемся начать сегодняшнее совещание с написания *пользовательских историй*. Это краткие заявления о функциях, но изложенные от лица пользователей. Например, «Как пользователь я хочу иметь возможность сохранить игру, которая еще не доиграна». Примерно так.

— Звучит довольно просто. А что мы будем делать с ними, когда сформулируем?

— Мы оценим каждую из них, приоритизируем, а потом найдем наилучший баланс между функциями и сроками, — сказал Карлос.

— Ну и как же составляются эти пользовательские истории? — спросил Фрэнк.

— Мы будем использовать вот эти карточки, — Карлос положил по пачке карточек перед каждым из присутствовавших. — Мне нравится записывать пользовательские истории в следующем формате. — Карлос взял фломастер и написал на магнитно-маркерной доске: «Как <тип пользователя> я хочу <цель> с тем, чтобы <причина>». — Делани, ты уже разобралась в этой игре Navannah, можешь дать нам пример?

— Конечно, — ответила Делани. — Начнем вот с этого: «Как игрок я могу отменить ход с тем, чтобы исправить серьезную ошибку».

— Это наша первая история? — спросил Прасад, тестирующий.

— Да, поэтому я запишу ее на карточке, чтобы не забыть, — сказала Делани и написала карточку истории (23.1.)

Как игрок я могу отменить ход с тем, чтобы исправить серьезную ошибку.

Карточка истории 23.1

— Теперь нам нужно оценить эту историю? — спросил Аллан.

— Пока нет, Аллан. Это будет легче сделать, когда у нас появится группа историй. Мы оценим их одновременно, — ответил Карлос.

— Теперь моя очередь. Если есть история об отмене действия, то нам нужна и история о повторе действия. «Как игрок я хочу иметь возможность повторить ход, который отменил, с тем, чтобы восстановить

последовательность ходов», — сказал Фрэнк.

— Хорошая история, Фрэнк. Запиши ее на карточку, — предложил Карлос менеджеру по продукту.

— А не лучше ли ввести это в электронную таблицу? — спросил Фрэнк.

— Возможно, позже. На совещании вроде нашего полезнее иметь реальные карточки. Вы увидите это, — сказал Карлос. — Они позволяют каждому из нас записать любую историю в любой момент. Нет необходимости ждать, пока кто-то введет ее с клавиатуры.

— А когда мы дойдем до планирования, карточки будут еще полезнее, — добавила Саша. — Мы сможем сортировать их по приоритетам и распределять по итерациям.

Фрэнк начал понимать преимущества использования карточек. Он мог написать новые карточки или забрать уже написанные. В электронной таблице проделать такое невозможно. Пока что этот новый «agile-процесс» был ему малопонятен, но уже увиденное внушало оптимизм. С растущим энтузиазмом Фрэнк написал карточку истории (23.2).

Как игрок я хочу иметь возможность повторить ход, который отменил, с тем, чтобы восстановить последовательность ходов.

Карточка истории 23.2

— Чтобы записать все истории, нужно... — начал было Карлос.

— Ты и в самом деле имеешь в виду все, Карлос? — перебила agile-тренера команды Саша.

— Да. В какой-то мере. Я хотел бы посвятить это совещание составлению как можно большего количества карточек историй. Отмена и повтор действия — это слишком мало. Потом, не нужно стремиться к тому, чтобы все наши истории были такими маленькими. На деле нам бы надо объединить эти две истории, отмену и повтор действия.

— Хорошо, я сделаю это, — сказал Фрэнк и написал карточку истории (23.3). — А как показать, что эта карточка связана с двумя другими? Их нужно как-то пронумеровать?

— Нет, просто разорви первые две карточки, — ответил Карлос. — Они нам больше не нужны.

Фрэнк разорвал первые две карточки.

Как игрок я хочу иметь возможность отменять и повторять ходы.

Карточка истории 23.3

— Карлос, Фрэнк не включил в карточку причину. Там нет «с тем, чтобы». Это нормально? — спросил Аллан.

— Эй, это всего лишь моя вторая история! — Фрэнк шутливо парировал выпад.

— В действительности, — пояснил Карлос, — «с тем, чтобы» требуется не всегда. Если это помогает сделать историю более ясной, то такое дополнение нужно. Но слишком беспокоиться о нем не стоит, особенно в истории вроде этой, где и так понятно, почему пользователь хочет иметь данную возможность.

— Карлос, ты сказал, что нам нужно написать все истории сегодня, даже если некоторые из них будут сформулированы в самом общем виде. Но разве нельзя добавить истории позже? — спросила Роуз.

— Конечно, можно. Мы обязательно будем добавлять какие-то истории позже. Если это будет не так, значит, мы подходим к игре недостаточно креативно. Я и мысли не допускаю, что позже у нас не появятся новые идеи, более удачные, чем те, с которых мы начинаем сегодня. Цель письменного изложения «всех» историй сегодня — описать наши знания, пусть даже в общем виде, о том, какие функции наиболее желательны. Это здорово поможет Саше и Аллану, которые будут разрабатывать архитектуру игры и продумывать подходы к созданию генератора ходов.

— Это здорово, Карлос. Продолжим. Может быть, мы все просто начнем писать карточки?

— Можно, — сказал Карлос. — Но лучше, если мы придадим этой работе определенную структуру. Давайте обсудим, что может понадобиться пользователю в различные моменты игры — до начала игры, во время игры и даже после ее окончания.

Со стороны членов команды послышались возгласы согласия, а Карлос продолжил:

— Что может пользователь захотеть сделать сразу после загрузки Navannah?

— Начать игру.

— Восстановить сохраненную партию.

— Выбрать уровень сложности.

— Ну вот, давайте запишем это, — сказал Карлос. Через несколько минут команда выдала кучу историй (табл. 23.1).

Таблица 23.1. Истории, сформулированные в результате обдумывания того, чего пользователи могут хотеть до начала игры

Текст истории
Как игрок я могу начать новую игру
Как игрок я могу восстановить сохраненную игру
Как игрок я могу выбирать уровень сложности
Как игрок я хотел бы иметь возможность использовать систему для игры с другим человеком на моем компьютере
Как игроку мне хотелось бы, чтобы оформление игры на экране было эстетически приятным
Как игрок я хотел бы иметь возможность выбирать между деревянной и металлической доской и фишками
Как новый игрок я хочу иметь возможность ознакомиться с интерактивным самоучителем по игре
Как игрок я хочу, чтобы в игре был музыкальный фон
Как игрок я могу выбирать музыкальный фон игры

— Карлос, — сказал Фрэнк, — если эти истории представляют то, что пользователь может захотеть до начала игры, почему в их число включили «Как игрок я хочу, чтобы в игре был музыкальный фон»? Ведь это нечто нужное игроку во время игры, а не до ее начала.

— Мы просто проводим мозговой штурм, чтобы определить круг пользовательских историй, — ответил Карлос. — Для нас пока не важно, какие истории связаны с периодом до начала игры, а какие с процессом самой игры. Такой подход, на мой взгляд, помогает осмыслить, что нам нужно в игре.

— Понятно. А что делать дальше?

— Теперь давайте подумаем, чего пользователь может захотеть во время игры.

В ответ на предложение Карлоса члены команды написали истории, представленные в табл. 23.2.

Таблица 23.2. Истории, сформулированные в результате обдумывания того, чего пользователи могут хотеть во время игры

Текст истории
Как игрок я хочу ставить фишку на доску с помощью клавиатуры или мыши
Как игрок я хочу видеть визуальный индикатор очередности хода
Как игрок я хотел бы видеть визуальный индикатор последней поставленной фишки (возможно, сделать ее мигающей)
Как игрок я хотел бы иметь возможность отменять и повторять ходы
Как игрок я хотел бы иметь возможность иногда пользоваться подсказками
Как игрок я хотел бы иметь возможность сохранять партии
Как игрок я хочу иметь возможность закончить игру
Как игрок я хочу возобновлять игру так, чтобы можно было отказаться от текущей игры и начать новую
Как новый игрок я хочу иметь доступ к онлайн-справочной системе
Как игрок я хочу, чтобы все фишки выигрышной фигуры мигали или подсвечивались с тем, чтобы можно было видеть выигрышную фигуру
Как новый игрок я хотел бы получать предупреждение о неудачном ходе и иметь возможность отменять его
Как новый игрок я хотел бы, когда наступает моя очередь делать ход, видеть клетку, на которую нужно поставить фишку, чтобы ее не занял компьютер и не обыграл меня
Как игрок я хотел бы, чтобы компьютеру требовалось на ход не более двух секунд на 2,0 ГГц РС

— А теперь, я думаю, нам надо перейти к тому, что пользователь может захотеть сделать после завершения игры, — сказал Фрэнк.

— Вряд ли после завершения игры ее участник может хотеть многого. Мы не подумали о возможности сохранения игры ранее, поэтому можно добавить такую историю, — сказал Прасад.

— Игрок может также захотеть вернуться и просмотреть ходы. Deer Black & White поддерживает такую возможность, — заметила Роуз. — Я могу сделать комментарий к ходу вроде такого: «Я также обдумывала возможность пойти на клетку A3».

— Окей, запишем это в виде историй, — сказал Карлос (результат представлен в табл. 23.3).

Таблица 23.3. Истории, сформулированные в результате обдумывания того, чего пользователи могут хотеть после игры

Текст истории
Как игрок я хочу, чтобы система отслеживала, сколько партий я выиграл, а сколько проиграл (в зависимости от уровня сложности?)
Как игрок я хотел бы добавлять комментарии к сохраненным партиям
Как игрок я хотел бы просматривать ходы с тем, чтобы иметь возможность анализировать сыгранные партии
Как игрок я могу сохранить партию

— Что дальше, Карлос? — спросил Аллан.

— Пока ты будешь шлифовать Deep Black & White в течение следующих двух недель, Делани проведет более глубокое исследование продукта.

— Да, — добавила Делани. — Я хочу проработать с потенциальными пользователями набор функций, которые они считают наиболее важными.

— Однако, прежде чем мы завершим сегодняшнюю работу, нам нужно дать каждой истории примерную оценку, — сказал Карлос.

— Конечно, давайте оценим их. Но сначала сделаем 10-минутный перерыв, — предложил Аллан.

Оценка пользовательских историй

— Теперь, — подвел итог совещания Карлос, — займемся оценкой размера историй в пунктах.

— А что такое *пункты*? — поинтересовался Фрэнк.

— Пункт — это безразмерная величина объема и сложности истории, — ответил Карлос. — Предположим, мы решили, что оценка сохранения партии составляет три пункта. Если, на наш взгляд, реализация восстановления сохраненной партии требует таких же усилий, то мы присвоим ей оценку «три пункта». Если же мы решим, что восстановление партии требует чуть меньше усилий, то дадим ей два пункта.

— Получается, что числа ничего не значат? — спросил Фрэнк.

— Только относительно. Два — вдвое больше одного. Восемь — вчетверо больше двух. Примерно так. Это все, что они значат, — пояснил Карлос. — Мы будем заниматься оценкой, играя в покер планирования. — Карлос раздал присутствующим по набору карт с цифрами 1, 2, 3, 5 и 8. — Теперь Делани прочтет нам одну из пользовательских историй. Мы обсудим ее и сможем задать вопросы. Затем каждый поднимет одну из розданных мною карт. Цифры на ней — это наши оценки. Все сначала выберут карты, а потом одновременно откроют их. Если цифры окажутся одинаковыми, это и

будет оценка, а если нет, то мы обсудим их и историю в течение нескольких минут и повторим процесс. Процесс повторяется до тех пор, пока все не придут к одной и той же оценке.

Карлос ответил на несколько вопросов относительно метода и попросил Делани прочитать первую историю: «Как игрок я могу начать новую игру».

— Пару первых историй всегда трудно оценивать, — сказала Саша. — Поскольку мы оцениваем истории по отношению друг к другу, тяжело, когда не с чем сравнивать. В принципе, нам надо просто решить, какой считать эту первую историю — маленькой или большой. Если после оценки четырех-пяти историй мы решим, что нужно изменить оценку первой истории, то это вполне можно сделать.

— Что ты имеешь в виду под «этой историей»? — спросил Аллан. — Пока неясно, что означает «начало новой игры».

— Мы написали эту историю, когда размышляли о том, чего пользователь может захотеть после запуска нашей программы, — сказала Делани. — Мы говорили, что он, скорее всего, захочет начать новую игру или восстановить старую. Вот и все, что имелось в виду.

— А эта история включает в себя построение чистого игрового поля и задание установок, с тем чтобы компьютер был готов играть? — спросил Аллан.

— На мой взгляд, включает. Не думаю, что для этой истории нужно красивое игровое поле. У нас есть другая история о создании эстетически привлекательного экрана. Но эта история, несомненно, требует создания игрового поля Navannah на экране.

Карлос немного подождал, чтобы удостовериться, нет ли других вопросов. «А теперь каждый выбирает карту с оценкой этой истории».

— Один, — сказал Аллан, показывая всем свою карту.

— Пока рано, Аллан, — сделал замечание Карлос. — У каждого должен быть шанс обдумать свое решение, не видя оценок, которые дают другие. Это позволяет избежать необъективности. — Он выдержал паузу: — Похоже, все сделали свой выбор. Открываем карты.

Цифры на картах варьировали от единицы у Аллана и Роуз до пяти у Прасада.

— Почему пять, Прасад? — спросил Карлос.

— На мой взгляд, эта история из разряда крупных, — ответил тот.

— И близко не валялась, — возразил Аллан. — Все, что нужно для нее, это пользовательский интерфейс, где игрок может выбрать кнопку «Начать новую игру» и получить чистое игровое поле Navannah. Поле не обязательно должно выглядеть красиво, а размещения фишек здесь не требуется. Для этого есть отдельные истории. Эта история простая.

— Хорошо, я не смотрел на нее под таким углом, — сказал Прасад.

Подождав немного, Карлос опять предложил всем выбрать карту с

оценкой. Теперь на большинстве карт была единица, и только Прасад выбрал «два». Прошел еще один короткий обмен мнениями, а потом Карлос поинтересовался, может ли Прасад согласиться с оценкой «один». Тот ответил, что может, и первую историю оценили в один пункт.

— Наша следующая история, — сказала Делани, — звучит так: «Как игрок я могу восстановить сохраненную игру».

После двухминутного обсуждения, как эта история должна работать, Карлос попросил членов команды выбрать карты. В первом раунде достичь согласия не удалось. Саша настаивала на том, что эта история в два раза больше, чем первая, поскольку предполагает создание чистого игрового поля Navannah, считывание сохраненной игры из файла и расстановку фишек на поле. Под влиянием аргументов Саши команда согласилась оценить историю в два пункта.

— Окей, следующая история — «Как игрок я могу выбирать уровень сложности». Давайте оценим ее, — сказала Саша.

— Это уже целый генератор ходов, — заметил Аллан. — Это довольно сложно. Можем ли мы на время оставить эту историю и вернуться к ней после того, как оценим несколько историй покрупнее?

— Я не против, — ответила Саша.

— Погодите, — сказала Роуз. — Я думаю, что это история о выборе пользователем уровня сложности, а не об игре компьютера на этом уровне сложности. Это просто — всего лишь одно или два поля на экране.

— Именно это я и имел в виду, когда записывал эту историю, — вставил слово Фрэнк.

— Отлично, но тогда нам нужна такая история: «Как игрок я могу играть против компьютера на разных уровнях сложности», — сказала Роуз.

— А можно разбить эту историю? — спросил Аллан. — Я думаю, она довольно сложна. Как насчет такой: «Как игрок я могу играть против слабого компьютерного противника»? И таких: «Как игрок я могу играть против сильного компьютерного противника» и «Как игрок я могу играть против среднего компьютерного противника»?

— Конечно, давайте так и поступим, — сказала Саша. — Начнем с выбора игроком уровня компьютерного противника. Выбирайте карты из ваших колод. Готовы? Открываемся. — Все подняли карты с цифрой «один». — На мой взгляд, это вполне резонно. Мы говорим, что у этой истории такой же размер, как и у «Как игрок я могу начать новую игру».

Все закивали в знак согласия с тем, что две истории действительно имеют один размер.

— Теперь перейдем к истории «Как игрок я могу играть против слабого компьютерного противника», — предложил Аллан, горевший желанием обсудить истории, связанные с генератором ходов, разрабатывать который, скорее всего, придется ему. — Насколько сильно должен играть наш слабый

движок?

— Он не может играть случайным образом, — сказала Делани. — Но он не должен быть очень сильным. Эта игра обманчиво проста, а на деле большинству людей требуется определенное время, чтобы выбрать хороший ход. Вместе с тем даже на низком уровне сложности необходимо знать, что лучше строить — кольцо, мост или вилку в зависимости от ходов игрока.

— Ну так давайте оценим ее, — сказал Аллан.

— Погодите, — вставил слово Прасад, — как мы будем тестировать это? Похоже, тестирование окажется сложным делом.

— Хороший вопрос, — сказал Карлос. — Какие будут соображения? — Он обвел взглядом присутствующих.

— Один из способов — идентифицировать группу позиций, посмотреть, что предлагает движок, и спросить у какого-нибудь хорошего игрока, насколько хороши эти ходы, — сказала Роуз.

— Звучит хорошо. А можно ли это автоматизировать? — спросила Делани. — Нам нужно автоматизировать тестирование так, чтобы, если движок начнет генерировать странные ходы, как наша Deep Black & White в апреле, мы сразу узнали об этом.

— Абсолютно согласен, — заметил Прасад. — Мы можем создать файл, который описывает позиции на доске и потом говорит, как компьютер должен реагировать.

— Приемлемых ходов может быть несколько, — сказал Аллан. — Этот файл должен допускать определение нескольких хороших ответов.

— Отличные ответы. Однако у нас нет необходимости браться за создание этого файла прямо сейчас. Давайте все же оценим историю, — вернул присутствовавших к теме Карлос.

В зале воцарилась тишина — все погрузились в обдумывание того, насколько сложной будет реализация этой истории. Выбрав карты, все одновременно подняли их.

— Аллан, ты ведь должен был поднять одну карту, а не все сразу. Тебе трудно принять решение?

— Нет, просто у меня не нашлось карты с достаточно большим числом, поэтому я и поднял их все. Все эти карты, — программист показал карты с цифрами 1, 2, 3, 5 и 8, — в сумме дают 19. Это, на мой взгляд, и есть правильная оценка.

— У меня есть карты и с большими числами — 13, 20, 40 и 100, — сказал Карлос и раздал всем дополнительные карты. — Я надеялся, что они не понадобятся нам, однако их тоже можно использовать.

— А почему ты надеялся, что они не понадобятся?

— По отношению к историям, которые мы оценивали до сих пор, любая история размером 100 пунктов будет слишком большой, чтобы уложиться в одну итерацию. То же самое будет, скорее всего, справедливо и для

остальных карт с большими числами, которые я вам только что раздал. В наличии нескольких крупных историй нет ничего страшного, и мы вполне можем оценить их, присвоив большие значения. Нужно, однако, помнить, что перед реализацией их придется разбивать на части, с тем чтобы они укладывались в одну итерацию. И еще нужно помнить, что оценки очень больших функций будут, скорее всего, менее точными. Именно по этой причине диапазоны становятся шире.

— Итак, Аллан, ты даешь этой истории 19. Уверен? — спросила Саша, глядя на восьмерку в своей руке.

— Абсолютно. Прежде всего мне нужно побольше узнать об этой игре. Движок должен будет распознавать, что пытается построить игрок — кольцо, мост или вилку. Он должен решить, какую фигуру ему нужно попытаться построить. Он должен решить, каким будет ход — наступательным или защитным. На создание даже первого слабого движка потребуются довольно много усилий.

— Ну, тебе виднее. Но 19 — слишком много, — сказала Саша.

— Прасад, почему ты считаешь, что должно быть три? — спросил Карлос, фокусируя внимание на оценщиках с предельными значениями.

— Тестирование не кажется мне сложным. Не думаю, что с ним придется много возиться. Я создам текстовые файлы для тестирования, и все.

— Конечно, но ведь мы оцениваем историю в целом, а не твою часть работы с нею. Тестирование, может быть, и тянет только на тройку, однако нужно учитывать и время, которое потребуется Аллану, — пояснил Карлос.

— Вот незадача! Тогда Аллан прав, — сказал Прасад. — Это большая история.

— Давайте посмотрим, прав ли он. Возьмите свои карты и дайте новую оценку с учетом того, что вы только что услышали, — предложил Карлос. Он выждал несколько секунд, чтобы у всех была возможность подумать. — Показывайте свои оценки.

На всех картах в этот раз было 20.

— Похоже, ты всех убедил, Аллан, — сказал Карлос. — Двадцать, однако, великовато для одной истории. Аллан, нельзя ли как-то разбить эту историю? Может быть, для начала достаточно слабого движка? С последующей доработкой перед поставкой?

— Не представляю, как это сделать, — сказал Аллан, задумался на мгновение и продолжил. — Можно, конечно, сделать так, чтобы движок каждый раз выбирал наступательный ход в стремлении построить свою фигуру независимо от ходов игрока. Но, на мой взгляд, это не очень хорошая идея.

— А что, если первый движок будет распознавать только кольца? — спросила Роуз. — Тогда мы на время смогли бы отбросить мосты и вилки. Аллан, разве ты не можешь написать программу, которая будет делать

наступательные и защитные ходы, ориентированные на построение — и блокировку — только колец?

— Без проблем. Это может сработать. Поэтому давайте разобьем нашу историю на три части по числу фигур.

Все согласились и оценили эти три истории, как показано в табл. 23.4.

Таблица 23.4. Три небольшие истории, получившиеся в результате разбивки эпопеи слабого движка

Текст истории	Оценка
Как игрок я могу играть против слабого движка, распознающего только кольца	8
Как игрок я могу играть против слабого движка, распознающего только мосты	5
Как игрок я могу играть против слабого движка, распознающего только вилки	8

— Когда это была одна история, мы ее оценивали в 20 пунктов. Теперь оценка составляет 21. Нужно ли снять один пункт с какой-нибудь истории, чтобы оценка осталась на уровне 20? — спросил Фрэнк.

— Нет. Пусть оценки остаются такими, как есть, — они не такие уж точные, — ответил Карлос. — Разбивка историй помогает нам видеть больше. Сумма оценок не обязательно должна быть равна оценке крупной истории.

— Аллан, а как мы поступим с движками среднего и высокого уровня? На твой взгляд, в каждом случае должна быть одна история или их тоже нужно разбить на кольца, мосты и вилки?

— Достаточно одной истории. Я думаю, нам нужно определить движок среднего уровня как такой, который никогда не допускает грубых ошибок, — сказал Аллан. — Он, возможно, не всегда будет делать лучшие ходы, но грубых ошибок точно не допустит. Тестирование можно организовать так, как мы уже говорили, и привлечь к этому процессу хороших игроков. Если он будет делать обоснованные с их точки зрения ходы, то все в порядке. Для движка высокого уровня можно добавить условие выбора наилучшего хода. В зависимости от результатов мы можем устанавливать, насколько далеко будет смотреть генератор ходов.

— Так как бы ты сформулировал это в виде историй? — спросил Фрэнк.

— Я думаю, так: «Как игрок я могу играть против движка среднего уровня», — сказала Делани.

— Пожалуй, — согласился Карлос. — Вы можете написать на этой карточке истории примечание о том, что «Всегда делает ходы, которые

хороший игрок считает приемлемыми» — это одно из условий удовлетворенности для данной истории. Оно поможет вам не забывать о том, что «движок среднего уровня» не требует создания еще одной истории о приемлемости ходов.

Команда продолжила обсуждение этой истории, провела раунд покера планирования и решила оценить ее в восемь пунктов.

— А потом, на мой взгляд, нам нужна история «Как игрок я могу играть против движка высокого уровня», — сказал Фрэнк.

— Она как минимум в два раза больше, чем история с движком среднего уровня, — заметил Аллан. — Предыдущую историю мы оценили в восемь. Но мне бы не хотелось давать этой истории оценку 16. Может быть, достаточно 13? Это самая скромная величина. Или все же остановимся на 16?

— Представьте, — сказал Карлос, — что карточки — это ведра. Если история равна 16, то она не войдет в ведро, которое вмещает 13. Введение новых величин нам ни к чему, поскольку это приводит к излишней точности. Мы не знаем, сколько здесь в точности: 13, 16 или 19. Важно помнить, что это всего лишь оценки, а не гарантии.

— Окей, я думаю, что эта история в два-три раза больше, чем движок среднего уровня, — сказал Аллан. — Это означает диапазон от 16 до 24. Поскольку 24 не войдет в ведро, вмещающее 20, значит, мне надо дать оценку 40? Это многовато. Наша история не тянет по размеру на пять историй с движком среднего уровня.

— Нет, я бы не стал делать этого, — ответил Карлос. — Это, конечно, ведра, но представьте, что их наполняют не водой, а песком. Значит, вы можете насыпать песок горкой.

— Перейдем к оценке, — сказал Аллан.

После этого обсуждения все единодушно согласились с Алланом и подняли оценку 20.

— Итак, мы оцениваем эту историю в 20, — сказал Карлос. — На данном этапе это нормально, но потом, вполне возможно, нам придется разбить такую работу как минимум на две части.

— Значит, нам не нужно разбивать эту историю на более мелкие, точно так же, как историю со слабым движком? — спросил Аллан.

— Нет, мы можем сделать это позже, когда подойдет очередь реализации этой истории, — ответил Карлос. — К тому времени мы будем знать больше, а разбивка прямо сейчас ничего полезного нам не принесет.

— Хорошо. Идем дальше, — предложила Делани. — На очереди история «Как игрок я хотел бы иметь возможность использовать систему для игры с другим человеком на моем компьютере». Имеется в виду, что два игрока могут сидеть за одним компьютером и делать ходы по очереди с помощью клавиатуры или мыши. Все, что нам здесь нужно, это получать от

компьютера сообщение, когда кто-то выигрывает.

— Другие функции, которые мы идентифицировали, по-прежнему работают, так ведь? — спросил Прасад. — Два игрока могут захотеть использовать функцию отмены и повтора или получать подсказку.

— Конечно, — ответила Делани.

— Думаю, нам надо добавить историю «Как игрок я хочу, чтобы компьютер распознавал выигрышную фигуру», — заметил Аллан.

— А разве это не часть историй о генераторе ходов?

— Да, но если мы вытащим ее, то сможем реализовать отдельно, а это позволит нам быстро написать версию игры «человек против человека» при наличии работающего генератора ходов.

Возражений не последовало, и новую историю оценили в два пункта.

— Аллан, ведь тогда понизятся оценки историй для слабого движка, не так ли? — спросил Фрэнк.

— Не должны. Я бы не стал менять их оценки. Но, если тебе так хочется, можно понизить восьмипунктовые истории до семипунктовых.

— Нет, не нужно этого делать, — вмешался Карлос. — В результате наши оценки окажутся слишком точными. Если бы мы хотели понизить оценку до пяти, то это было бы нормально, а так оставим ее равной восьми.

— Пять — это слишком мало, — сказал Аллан.

— Итак, возвращаемся к оценке истории «Как игрок я хотел бы иметь возможность использовать систему для игры с другим человеком на моем компьютере».

После двух раундов покера планирования члены команды пришли к согласованной оценке: три пункта.

— Теперь очередь истории «Как игроку мне хотелось бы, чтобы оформление игры на экране было эстетически приятным», — прочитала Делани.

— Ну наконец-то моя история, — сказала Роуз, художник.

— Да, только мне хочется переформулировать ее, — сказала Саша. — В нынешней форме она расплывчата и очень большая. Я бы предпочла, чтобы она выглядела так: «Как игрок я хотел бы иметь возможность выбирать между деревянными доской и фишками и металлическими доской и фишками». Что еще мы подразумевали под словами «эстетически приятный»?

— Это имеет отношение к общему облику и впечатлению от игры, — ответил Фрэнк. — Скорее всего, нам не нужны многочисленные меню, но они должны выглядеть красиво. Опции меню должны иметь логичное расположение. Нам нужен привлекательный начальный экран при загрузке игры. Доска и фишки — это весь пользовательский интерфейс. Наверное, нужно сделать какой-нибудь фон вокруг игрового поля. Думаю, это должно быть что-то художественное. Роуз нужно время, чтобы сделать нечто

оригинальное.

— Понятно. А нельзя ли разбить это на отдельные истории? — спросила Саша.

— Можно, — вставила слово Делани, — но все может оказаться не так просто. На мой взгляд, меню нужно разрабатывать в контексте тех функций, для которых требуются эти меню или их разделы. Роуз, тебя устроит, если будет одна история для начального экрана и еще одна история для художественного фона?

— Да, вполне. Мне в любом случае пришлось бы работать над ними по отдельности.

Команда быстро оценила следующий набор историй.

— А что нам делать с этой историей? — спросил Прасад, указывая на карточку, где значилось «Как игрок я хотел бы иметь возможность иногда пользоваться подсказками».

— Что ты имеешь в виду?

— Если мы сделаем генератор ходов, эта история будет несложной. Нужно лишь, чтобы генератор предложил новый ход, но не компьютеру, а игроку. Это проще простого. Но если у нас не будет готового генератора ходов, то история становится по размеру такой же, как и создание целого генератора ходов.

— Ну вот, мы получили взаимозависимость: подсказку можно сделать только после реализации генератора ходов, — сказал Аллан.

— Да, но я не думаю, что это будет проблемой, — заметила Саша. — Разработка генератора ходов до функции подсказки вполне естественна для реализации этих историй. Взаимозависимость не будет нам мешать, и на нее не нужно обращать внимания. А чтобы не забыть о существовании такой взаимозависимости, ее можно записать в карточку. Так или иначе, эту историю можно оценить, исходя из предположения, что генератор ходов существует.

Все согласились, и историю оценили в один пункт.

— А что бы мы сделали, если бы и впрямь захотели реализовать подсказку до разработки генератора ходов?

— Иногда невозможно избавиться от взаимозависимости и с этим нужно смириться, как вы это делали до использования agile-подхода, — пояснил Карлос. — Чаще всего, однако, от нее удастся избавиться. В нашем случае можно написать код, который позволяет пользователю запрашивать подсказку, отображать ее на экране и делать ход за пользователя, если он согласен. Чтобы получить подсказку, в системе должно быть нечто, называемое генератором подсказок. В конечном итоге он должен будет обращаться к генератору ходов для получения рекомендации, однако сейчас можно ограничиться тем, чтобы он выдавал ближайшую свободную клетку или случайную свободную клетку. Таким образом можно реализовать

историю подсказки даже до начала разработки генератора ходов.

— Надо не забыть вернуться к этому вопросу позднее и сделать так, чтобы генератор подсказки действительно обращался к генератору ходов, — сказал Аллан.

— Да, — подтвердил Карлос. — Мы можем создать еще одну историю, что-нибудь вроде такой: «Как игрок я хочу получать хорошие подсказки», а не просто подсказки.

— Это, скорее всего, очень маленькое изменение, — заметил Аллан. — Мы удаляем то, что генератор подсказки делает для поиска свободной клетки, и вместо этого посылаем запрос генератору ходов. Такую историю нужно оценить как единицу — точно так же, как и первоначальную историю. Не похоже, чтобы это давало какой-то выигрыш.

— Ну да, — сказал Карлос, — иногда полезно иметь карту с нулем вместо значащих цифр, которые у нас есть. Поскольку нам нужно вернуться к генератору подсказок и подкорректировать код, требуется карточка истории, не позволяющая забыть об этом. Однако задача настолько проста, что хочется дать ей нулевую оценку.

Карлос раздал всем игрокам в покер планирования карты с нулем: «Я придержал их с тем, чтобы вы не использовали ноль до тех пор, пока не зайдет разговор о том, какая история может быть нулевой». У каждого участника теперь была колода карт, содержащая 0, 1, 2, 3, 5, 8, 13, 20, 40 и 100.

— В нашем случае, впрочем, необходимо добавить поддержку подсказки после реализации генератора ходов. Так? — спросил Фрэнк.

— Да, мы оформим это как однопунктовую историю, — ответила Саша.

Обсуждение продолжалось до тех пор, пока команда не оценила все записанные истории. Эти истории с оценками приведены в табл. 23.5.

— Так что мы будем делать после этого? Нам нужна еще одна двухнедельная итерация на Deep Black & White, прежде чем мы сможем отправить ее издателю, — сказал Фрэнк.

— Пока остальные работают над ней, я опрошу нескольких потенциальных покупателей новой игры, — сказала Делани. — Я хочу понять, какие функции наиболее важны.

— На это уйдет много времени?

— Нет, я собираюсь закончить все до завершения итерации, — добавила Делани. — Давайте соберемся опять через две недели, сразу после отгрузки Deep Black & White и после того, как Фрэнк соберет нас, чтобы отпраздновать завершение этой работы.

— Хорошая идея. Я займусь организацией этого мероприятия. Гамбургеры подойдут, Делани? Или мы замахнемся на пиццу?

Таблица 23.5. Оценки всех записанных историй

Текст истории	Оценка
Как игрок я могу начать новую игру	1
Как игрок я могу восстановить сохраненную игру	2
Как игрок я могу выбирать уровень сложности	1
Как игрок я могу играть против слабого движка, распознающего только кольца	8
Как игрок я могу играть против слабого движка, распознающего только мосты	5
Как игрок я могу играть против слабого движка, распознающего только вилки	8
Как игрок я могу играть против движка среднего уровня	8
Как игрок я могу играть против движка высокого уровня	20
Как игрок я хотел бы иметь возможность использовать систему для игры с другим человеком на моем компьютере	3
Как игрок я хочу, чтобы компьютер распознавал выигрышную фигуру	2
Как игрок я хочу видеть привлекательный начальный экран при загрузке игры	5
Как игрок я хочу видеть привлекательный фон вокруг игрового поля, сочетающийся с выбранной доской	5
Как игрок я хотел бы иметь возможность выбирать между деревянной и металлической доской и фишками	8
Как игрок я хотел бы иметь возможность иногда пользоваться подсказками	1
Как новый игрок я хочу иметь возможность ознакомиться с интерактивным самоучителем по игре	8
Как игрок я хочу, чтобы в игре был музыкальный фон	5
Как игрок я могу выбирать музыкальный фон игры	1
Как игрок я хочу ставить фишку на доску с помощью клавиатуры или мыши	3

Текст истории	Оценка
Как игрок я хочу видеть визуальный индикатор очередности хода	2
Как игрок я хотел бы видеть визуальный индикатор последней поставленной фишки (возможно, сделать ее мигающей)	2
Как игрок я хотел бы иметь возможность отменять и повторять ходы	2
Как игрок я хотел бы иметь возможность сохранять партии	3
Как игрок я хочу иметь возможность закончить игру	1
Как игрок я хочу возобновлять игру так, чтобы можно было отказаться от текущей игры и начать новую	1
Как новый игрок я хочу иметь доступ к онлайн-справочной системе	8
Как игрок я хочу, чтобы все фишки выигрышной фигуры мигали или подсвечивались с тем, чтобы можно было видеть выигрышную фигуру	3
Как новый игрок я хотел бы получать предупреждение о неудачном ходе и иметь возможность отменять его	8
Как новый игрок я хотел бы, когда наступает моя очередь делать ход, видеть клетку, на которую нужно поставить фишку, чтобы ее не занял компьютер и не обыграл меня	3
Как игрок я хотел бы, чтобы компьютеру требовалось на ход не более двух секунд на 2,0 ГГц РС	8
Как игрок я хочу, чтобы система отслеживала, сколько партий я выиграл, а сколько проиграл (в зависимости от уровня сложности?)	3
Как игрок я хотел бы добавлять комментарии к сохраненным партиям	3
Как игрок я хотел бы просматривать ходы с тем, чтобы иметь возможность анализировать сыгранные партии	5

Подготовка к исследованию продукта

Тихим ранним утром следующего дня Делани устроилась за столом в новом помещении команды и занялась подготовкой анкеты для опроса потенциальных покупателей игры. Фрэнк договорился с ней о встрече в

течение дня, чтобы обсудить, как она собирается приоритизировать функции. Как менеджеру по продукту, Фрэнку нужно было принять окончательное решение о функциональном наполнении продукта, и Делани знала, что он в значительной мере полагается на ее исследование. Ей хотелось как следует подготовиться к встрече с ним. К тому моменту, когда появился Фрэнк, Делани почти закончила работу с анкетой.

— Здравствуй, Делани, — поприветствовал Фрэнк аналитика. — У тебя найдется для меня минутка в это чудесное утро?

— Конечно. Я пришла пораньше, чтобы доработать анкету, которую собираюсь разослать. Посмотри, что получилось.

— Давай посмотрим, — сказал Фрэнк, усаживаясь в кресло. — Покажи, что у тебя получилось.

— Я напечатала это полчаса назад, — сказала Делани и протянула Фрэнку страницу с текстом (табл. 23.6). После этого я добавила еще кое-что, но это все равно дает представление о том, что я собираюсь разослать.

Таблица 23.6. Начало анкеты Делани

Как вы отнесетесь к...	Мне это нравится	Я ожидал, что это будет выглядеть так	Мне безразлично	Я могу смириться с этим	Мне это не нравится
Наличию возможности выбрать игру с довольно слабым компьютерным противником?					
Отсутствию возможности выбрать игру с довольно слабым компьютерным противником?					
Наличию возможности выбрать игру с компьютерным противником средней силы?					
Отсутствию возможности выбрать игру с компьютерным противником средней силы?					
Наличию интерактивного руководства, которое помогает освоить правила игры?					
Отсутствию интерактивного руководства, которое помогает освоить правила игры?					

Фрэнк просмотрел страницу, которую дала ему Делани, и спросил: «Ты задаешь каждый вопрос дважды. Зачем?»

— Первую форму вопроса называют функциональной, а вторую —

дисфункциональной, поскольку она подчеркивает, что функция отсутствует. Наличие вопроса в двух формах позволяет получить более четкое представление об отношении человека, чем в случае одного вопроса типа «Насколько вам нужна эта функция?». Такой подход однозначно показывает, как пользователь отнесется к наличию функции и к ее отсутствию.

— Можешь ли ты привести пример, показывающий, насколько это может быть важным?

— Конечно. Допустим, мы спрашиваем «Как вы отнесетесь к наличию функции отмены и повтора ходов?» и пользователь говорит, что ожидает наличия такой возможности. Тогда мы спрашиваем «Как вы отнесетесь к отсутствию функции отмены и повтора ходов?» и пользователь говорит, что ему это не нравится. Это указывает на то, что данная функция обязательна для пользователя. Он ожидает наличия возможности отмены и повтора; ему не нравится отсутствие такой возможности.

— Окей, согласен.

— Теперь предположим, что мы задаем тому же пользователю вопрос: «Как вы отнесетесь к наличию интерактивного справочника, помогающего освоить правила игры?» Ему нравится такая идея. Мы также задаем этот вопрос в дисфункциональной форме: «Как вы отнесетесь к отсутствию интерактивного справочника, помогающего освоить правила игры?» — а он отвечает, что ему безразлично. В этом случае функцию называют привлекательной. Мы нашли функцию, которую пользователь не ожидает получить или с отсутствием которой он может смириться. Однако он говорит, что эта функция ему нравится. Таким образом, мы узнаем, что данная функция необязательная, но из-за ее привлекательности пользователь может дополнительно заплатить за нее.

— Но разве это нельзя выяснить, попросив пользователей оценить важность функций по шкале от одного до пяти, как мы сделали в случае Deer Black & White? — спросил Фрэнк.

— Не совсем, — ответила Делани. — Одна оценка по шкале вроде этой просто скажет нам, насколько пользователь ценит функцию. Она не позволит определить, как сильно пользователь отреагирует на отсутствие этой функции. Я бы не оценила колеса слишком высоко, если бы ты спрашивал меня о функциях автомобиля, но пришла бы в ярость, если бы ты открутил у него колеса.

— Кому ты собираешься давать эту анкету?

— Я собираюсь разослать ее примерно 500 членам ряда игровых клубов, в которых, насколько мне известно, играют в Havannah. Многие из них находятся в Европе и в крупных американских городах. Тем, кто ответит, мы предлагаем купон на скидку в размере \$10 при покупке наших существующих игр. Я рассчитываю получить порядка полусотни ответов. А

еще я собираюсь провести небольшой телефонный опрос.

— Не многовато ли будет вопросов для респондентов, если о каждой пользовательской истории спрашивать дважды? — поинтересовался Фрэнк.

— Не думаю. Я сгруппирую некоторые пользовательские истории в *темы*. Например, у нас есть две истории, касающиеся фоновой музыки. Их вполне можно объединить в тему о фоновой музыке. Я объединю истории о художественном фоне и начальном экране в одну тему об эстетически приятном впечатлении от игры. И, конечно, не буду спрашивать о сохранении сыгранных партий и восстановлении сохраненной игры. Мы и так знаем, что это обязательные функции.

— Отлично. Я хочу убедиться в том, что это необременительно. Как предполагается использовать ответы, которые ты получишь?

— Я хочу разделить функции на три категории. Первая — это функции, которые обязательно должны быть. В их число войдут такие функции, как сохранение и восстановление партий. Вторая категория — это такие функции, которых чем больше, тем лучше. На мой взгляд, сюда относится функция выбора уровня сложности. Чем больше уровней — например, высокая сложность, средняя сложность и низкая, — тем лучше. Третья категория — привлекательные функции. К ним относятся такие функции, которые большинство пользователей не ожидают, но, когда видят, начинают хотеть их. Правильное сочетание функций из последних двух категорий плюс все обязательные функции могут сложиться в очень привлекательный продукт.

Прежде чем Фрэнк успел ответить, к нему с Делани подкатил на своем кресле Аллан.

— Привет, — сказал он. — Я невольно подслушал часть вашего разговора. Звучит действительно интересно. А у остальных членов команды будет доступ к результатам твоего опроса?

— Конечно, Аллан.

— Чудесно. Знаешь, мне было бы интересно послушать парочку твоих телефонных интервью, хочу знать, что говорят пользователи. Это осуществимо? — спросил Аллан.

— Без проблем, — ответила Делани. — Понимание нашей аудитории идет на пользу всей команде. Чем больше ты и другие технари хотят участвовать в этом, тем лучше для меня. Я буду больше знать о ваших представлениях, а вы узнаете, о чем говорят клиенты.

— Идет, — сказал Аллан. — Сегодня я не могу сидеть на телефоне, мне нужно устранить ошибки в системе подсчета очков в Deer Black & White. Но в остальные дни этой недели дай мне знать, когда соберешься обзванивать пользователей.

— Непременно, — ответила Делани.

— У тебя, похоже, серьезные планы, — заметил Фрэнк. — Очень

интересно. Можно рассчитывать на какие-нибудь результаты на следующей неделе?

— Несомненно. ИТ-группа обещает разместить анкету на нашем сайте уже завтра. После этого предварительные результаты могут появиться в любой момент.

— Чудесно. Спасибо тебе за информацию, — сказал Фрэнк и откланялся.

Планирование итерации и релиза, раунд 1

Через две недели после совещания, посвященного формулированию и оценке пользовательских историй, команда вновь собралась в конференц-зале. К 9:00 пришли все, даже Роуз, которая стала отвозить Брук в школу на пять минут раньше, чтобы успевать в офис к началу ежедневных летучек.

— Сегодня у нас три цели, — начал Карлос. — Прежде всего мы должны спланировать вашу первую двухнедельную итерацию. После этого Делани расскажет, что она узнала в результате исследования пожеланий потенциальных покупателей. Наконец, нам нужно сделать первый очень грубый набросок плана релиза и календарного графика. Вопросы есть?

Саша выждала несколько секунд, чтобы понять, есть ли вопросы. Вопросов не было.

Планирование первой итерации

— Так с чего мы начнем разработку? — спросила Роуз.

— Я хочу начать с генератора ходов, — сказал Аллан. — Это наш самый большой источник риска.

— У меня возражений нет, — заметила Саша, другой программист команды. — Чем я могу помочь, Аллан? Обычно все, что касается искусственного интеллекта, твоя доля работы.

— Да, но кое-что можешь сделать ты. Спасибо, — ответил Аллан. — Твоя помощь очень кстати, поскольку меня беспокоит сложность разработки сильного движка.

— Так давайте начнем с истории «Как игрок я могу играть против слабого движка, распознающего только кольца». Ты ведь это хотел сделать в первую очередь, Аллан? — спросила Делани.

— Да. Первое, что нам нужно сделать, это закодировать классы, которые будут отслеживать состояние доски, следить за очередностью ходов и т.п. Саша, ты здесь определенно можешь помочь.

— Окей. Сколько времени, на твой взгляд, это займет? Пару дней?

— Пару полных дней. Лучше сказать, 16 часов, — ответил Аллан.

— Аллан, давай будем записывать каждую задачу на отдельной карточке. Напиши это на карточке и укажи 16 в углу, чтобы не забыть оценку, — сказал Карлос.

Аллан заполнил карточку задачи (23.1).

*Кодирование классов управления состоянием.
16 часов.*

Карточка задачи 23.1

— Может быть, поставить на ней мои инициалы, чтобы не забыть, чья это карточка?

— Нет. Иначе будет казаться, что этим занимаешься ты один. Лучше, если мы на текущем этапе не будем распределять работу между исполнителями.

— Мне нужно будет протестировать это, — сказал Прасад. — Времени на такую задачу потребуется немного, но это первый код нынешнего проекта, и я хочу удостовериться, что все делаю правильно. Я хочу сказать, что тестирование займет 10 часов.

Он заполнил карточку задачи 23.2.

*Написать автоматизированные тесты для классов управления состоянием.
10 часов.*

Карточка задачи 23.2

— Карлос, в прошлый раз ты говорил, что мне нужно разбить работу на небольшие части, — сказал Аллан. — Я ведь не могу сказать, что разработка генератора ходов для колец займет две недели, так ведь?

— Да, не можешь. При идентификации задач нам нужно держать их в пределах от одного до 16 часов. В идеале следует стремиться к тому, чтобы выполнять по одной задаче каждый день; иначе говоря, на них в среднем должно уходить пять-шесть часов, поскольку у нас всегда есть другие повседневные задачи.

— В таком случае, — произнес Аллан, — я сначала сделаю генератор ходов, который сможет разыгрывать шесть шестиугольников подряд, чтобы составить кольцо. Противника, пытающегося помешать генератору, не будет. Генератор начнет ходить со случайной клетки и пытаться построить кольцо за шесть ходов. Потом я доработаю его так, чтобы он мог строить кольца, даже если противник будет пытаться блокировать их. После этого я

переключусь на защитную стратегию и попробую сделать генератор, который научится блокировать действия игрока, пытающегося построить кольцо. Думаю, для этой истории этого хватит.

— Тогда я пишу такую карточку задачи: «Создать генератор ходов, который может построить кольцо в отсутствие блокировок». Именно это ты собираешься делать в первую очередь, — сказала Саша.

— Пожалуй, на это у меня уйдет львиная доля дня. Поставь, пожалуйста, на карточке шесть часов, — попросил Аллан.

— Аллан, без обид, ты всегда чересчур оптимистичен перед началом разработки нового генератора ходов, — заметила Роуз, художник.

— Да, есть такое. Давайте лучше удвоим оценку, — согласился Аллан.

Саша зачеркнула «шесть» на карточке и поставила 12.

Карлос руководил обсуждением оценок остальных задач, идентифицированных Алланом. «Есть ли еще неидентифицированные задачи для этой истории?» — спросил он.

— Не забывайте, что нужно еще время на тестирование! — сказал Прасад. — Если Аллан будет давать мне код после выполнения каждой из идентифицированных им задач, то я смогу определять и автоматизировать тесты одновременно с ним. О выявленных ошибках он будет узнавать, пока информация о коде еще не выветрилась у него из головы. Я написал задачи на карточках, пока Аллан рассказывал, как он будет работать, но они еще не оценены. Может быть, сделаем это вместе?

Когда члены команды закончили обсуждение истории «Как игрок я могу играть против слабого движка, распознающего только кольца», они представили карточки задач с оценками (табл. 23.7).

Таблица 23.7. Задачи с оценками для истории «Как игрок я могу играть против слабого движка, распознающего только кольца»

Задача	Часы
Кодирование классов управления состоянием	16
Написать автоматизированные тесты для классов управления состоянием	10
Создать генератор ходов, который может построить кольцо в отсутствие блокировок	12
Написать автоматизированные тесты для построения кольца в отсутствие блокировок	12
Создать генератор ходов, который может построить кольцо, даже если игрок пытается заблокировать его	8
Идентифицировать тесты для построения кольца в случае блокирования	4
Автоматизировать тесты для построения кольца в случае блокирования	4
Создать генератор ходов, который может заблокировать попытки игрока построить кольцо	12
Идентифицировать тесты для блокирования попытки игрока построить кольцо	4
Автоматизировать тесты для блокирования попытки игрока построить кольцо	2

— В этой итерации порядка 84 часов. Работа распределена между Алланом, Сашей и Прасадом, — сказал Карлос. — Исходя из того, что я видел здесь в течение двух дней, думаю, нам следует рассчитывать на шесть часов продуктивной работы в день на человека. Оставшаяся часть дня уходит на переписку, переговоры с другими проектными командами, ежедневные совещания и т.п. плюс полдня каждые две недели, затрачиваемые на планирование.

— Шесть, на мой взгляд, нормально.

— На мой тоже. Может быть, даже ближе к пяти, но я постараюсь, чтобы в этом проекте было шесть. Думаю, это реально, если отказаться от некоторых мелочей, которые каждый день съедают время, — сказала Саша.

— Теперь, когда вы разбили первую историю на задачи, — сказал Карлос, — вам нужно задаться вопросом, готовы ли вы взять обязательство реализовать эту историю.

— Историю или задачи? — спросил Фрэнк. — В отношении чего мы должны взять обязательство?

— Хороший вопрос, Фрэнк, — заметил Карлос. — Вы берете обязательство в отношении истории. Задачи идентифицировались как способ определить, какой объем работы предстоит сделать, и помочь взять обязательство. Поскольку обязательство берется в отношении истории, а не задач, важно, чтобы вы думали о нем как об обязательстве всей команды. Я не хочу, чтобы в конце итерации Саша сказала, что она сделала свою часть

работы, связанной с классами управления состоянием, а Аллан нет. Это не приводит ни к чему хорошему. Вы можете думать о задачах, но обязательства принимаете в отношении историй.

— Любопытно, — сказал Фрэнк. — Мне нравится различие.

— Я могу взять обязательство в отношении этой истории, — заявил Прасад. — У меня задачи тестирования объемом 36 часов. При шести часах в день в течение 10 дней я располагаю 60 часами.

— А у нас с Алланом объем задач 48 часов. Это не будет проблемой, — сказала Саша.

— Отлично. Мы спланировали первую историю и взяли обязательство реализовать ее, — сказал Фрэнк. — На очереди вторая по важности история.

— Фрэнк, прежде чем сделать это, давайте посмотрим на доступность членов команды в течение следующих двух недель, — предложил Карлос. — Мы решили исходить из шести часов в день на человека, но может оказаться, что у кого-то на период итерации приходится выходной или отпуск.

— Я беру два выходных дня. Я отправила пару картин на выставку в Сан-Франциско и хочу посмотреть, как там идут дела, — сказала Роуз.

— Пожалуй, и я возьму выходной. В пятницу или в понедельник, — вставил слово Аллан.

— А ко мне на следующей неделе приезжают родители. Слишком длительное общение с ними сведет меня с ума, поэтому я буду работать сверхурочно! — пошутила Делани.

Пока шел разговор, Карлос делал на доске маркером пометки (табл. 23.8). «Это поможет нам определиться, какой объем обязательств мы можем взять, когда будем рассматривать следующие истории», — пояснил он.

Таблица 23.8. Кто будет отсутствовать во время итерации

Имя	Отсутствие
Фрэнк	
Аллан	1 день?
Роуз	2 дня
Саша	
Прасад	
Делани	

— Спасибо, Карлос, — сказал Фрэнк. — Что теперь на очереди?

— Мне очень хочется получить возможность игры человека с человеком, — сказала Делани.

— Это хорошая идея, — согласился Аллан. — Она потребует от нас запрограммировать логику распознавания выигрышной позиции. К этому можно легко добавить функции отмены и повтора, а я тем временем сделаю подходящую первую версию генератора ходов.

— Давайте разобьем историю «Как игрок я хотел бы иметь возможность использовать систему для игры с другим человеком на моем компьютере» на задачи, — сказал Фрэнк.

Все согласились. Полученный в результате обсуждения перечень задач представлен в табл. 23.9.

Таблица 23.9. Задачи с оценками для истории «Как игрок я хотел бы иметь возможность использовать систему для игры с другим человеком на моем компьютере»

Задача	Часы
Очень простая доска и графика	4
Нарисовать пустую доску	2
Щелчок по шестигранной клетке добавляет новую фишку соответствующего цвета	4
Компьютер знает, когда фишки образуют выигрышную фигуру	6
Спроектировать тесты	6
Автоматизировать тесты	8

— Ну, как команда вы можете взять обязательство по реализации этой истории в дополнение к первой? — спросил Карлос.

— Погодите, разве эта история не включает в себя другую историю? Мы ведь записали: «Как игрок я хочу, чтобы компьютер распознавал выигрышную фигуру» и включили это как часть в игру против человека, — сказал Прасад. — Может, стоит удалить эту историю?

— Я пока не знаю. Мы включили ее случайно. Давайте посмотрим, сможем ли мы принять обязательство выполнить работу, которую только что идентифицировали. Если не сможем, то у нас будет возможность удалить распознавание выигрышных фигур и оставить вопрос об окончании игры на усмотрение игроков, — предложила Делани. — Как вы считаете? Сможем ли мы взять обязательство по реализации этой истории?

— Да, — почти в один голос сказали Аллан и Саша.

— На мой взгляд, тоже, — высказалась Роуз.

— Не уверен, — возразил Прасад.

— Поясни свой вывод, Прасад, — попросил Карлос тестировщика.

— На две истории мне нужно 50 часов. Конечно, как мы уже говорили, в моем распоряжении 60 часов, но объем работы кажется мне слишком большим. Нужно посмотреть, как все пойдет.

— Не забывай, Прасад, что задачи, связанные с тестированием, поручаются не именно тебе, — сказал Карлос. — Ими может заниматься любой. Если ты будешь зашиваться, к задачам тестирования подключатся другие.

— Совершенно верно. Я обычно выполняю некоторые задачи тестирования, — заметила Роуз. — Мне хотелось бы начать разработку дизайна доски в этой итерации, но это подождет, если нужно будет переключиться на тестирование.

— Спасибо, Роуз, — сказал Прасад. — Даже без помощи я думаю, что смогу взять обязательство по реализации этой истории. Однако я уже приближаюсь к пределу.

Команда взялась за обсуждение следующей истории: «Как игрок я могу играть против слабого движка, распознающего только мосты». Идентифицированные задачи и их оценки представлены в табл. 23.10.

Таблица 23.10. Задачи с оценками для истории «Как игрок я могу играть против слабого движка, распознающего только мосты»

Задача	Часы
Движок может находить путь от одного угла к другому (т. е. строить мост)	4
Идентифицировать и автоматизировать тесты для моста простой формы	6
Движок может при строительстве моста обходить препятствия	12
Идентифицировать и автоматизировать тесты для моста с обходом препятствий	4
Движок знает, когда отказаться от строительства конкретного моста	8
Идентифицировать тесты для отказа от строительства моста	4
Автоматизировать тесты для отказа от строительства моста	2
Движок пытается помешать игроку построить мост	4
Идентифицировать тесты для блокирования моста, выстраиваемого игроком	4
Автоматизировать тесты для блокирования моста, выстраиваемого игроком	4
Движок может выбирать между ходом, ведущим к созданию моста, и ходом, ведущим к созданию кольца	16
Идентифицировать тесты для выбора между строительством моста и кольца	6
Автоматизировать тесты для выбора между строительством моста и кольца	2

— Можем ли мы взять обязательство по реализации и этой истории? — спросила Роуз.

Фрэнк заметил и оценил то, что кто-то из команды спрашивает о готовности всех взять обязательство. Это нравилось ему намного больше, чем постоянно задавать вопрос самому, как менеджеру по продукту: «Вы готовы взять обязательство?»

— Пока мы разговаривали, я добавила вот это, — сказала Саша, поднимаясь с кресла. — Смотрите, что получилось. Она нарисовала табл. 23.11.

Таблица 23.11. Итоговое количество часов для трех историй

История	Часы, связанные с программированием	Часы, связанные с тестированием	Часы, связанные с художественным оформлением
Как игрок я могу играть против слабого движка, распознающего только кольца	48	36	0
Как игрок я хотел бы иметь возможность использовать систему для игры с другим человеком на моем компьютере	12	14	4
Как игрок я могу играть против слабого движка, распознающего только мосты	44	38	0
Всего	104	88	4

— Не знаю, как остальные, а я-то уж точно могу взять обязательство на четыре часа в течение следующих двух недель, — пошутила Роуз.

— У нас два программиста, и у каждого есть 30 часов в неделю и двухнедельная итерация; я думаю, с нашей стороны все окей, — сказала Саша.

— Согласен, — поддержал ее Аллан.

— Впрочем, братъ на себя что-то дополнительно к этому очень не хотелось бы, — добавила Саша. — А кроме того, объем тестирования слишком велик для одного Прасада.

— Я собираюсь помочь ему, — сказала Роуз.

— Я тоже, — добавила Делани. — Я могла бы заняться автоматизацией, но могу помочь и в идентификации хороших тестов.

— С такими помощниками я вполне управлюсь, — сказал Прасад. — Думаю, мы можем взять обязательство в отношении этого.

— Итак, все согласны? — спросил Фрэнк.

Возражений не последовало.

— Я хотела бы добавить несколько задач в итерацию, просто для уверенности в том, что мы учитываем их, — сказала Делани. — Мне нужно опросить еще нескольких игроков. Мы еще не говорили о моем исследовании продукта, но во время обсуждения всплыли некоторые моменты, которые мне хотелось бы уточнить. Если бы мы добавили порядка 12 часов для меня на дополнительное исследование продукта, то было бы здорово. Это не мешает мне посвящать большую часть своего времени тестированию. — А еще я хотела бы сделать предварительные наброски вида доски и фишек, чтобы их можно было разослать. Для получения обратной связи всегда требуется определенное время. Я хочу нарисовать четыре варианта досок и фишек, и мне нужно по четыре часа на каждый из них.

— На мой взгляд, это логично, — сказал Фрэнк. — Давайте добавим эти задачи.

— Итак, если суммировать все перечисленное, готовы ли мы принять обязательство по реализации этого? — спросила Саша. — Теперь в итоге получается 224 часа. В проекте участвуют пять человек — Аллан, Прасад, Роуз, Делани и я — это около пяти часов в день на человека. Но это лишь в том случае, если бы все работы и мы были взаимозаменяемыми. К сожалению, это не так.

— А кроме того, — добавил Карлос, — лучше добавить что-нибудь в середине итерации, чем выбросить что-то за недостатком времени.

Все согласились с тем, что уровень трудозатрат такой, как надо, — не слишком велик, не слишком мал, просто такой, как надо. Все обязались как команда реализовать три названные истории за две недели.

— Фантастика! — сказал Фрэнк. — Чудесно сознавать, что мы достигнем видимого прогресса так быстро. Обычно требуется намного больше времени, чтобы получить что-нибудь пригодное для показа.

— Давайте сделаем 15-минутный перерыв, — предложил Карлос. — Когда мы вернемся, я хочу перейти ко второму вопросу нашего совещания: планированию релиза. Нам нужно сделать первоначальные прикидки, сколько времени займет проект.

Планирование релиза

Когда все вернулись и расселись за столом, Фрэнк начал совещание:

— Делани, расскажи нам об исследовании продукта и о его результатах.

— Вот что я сделала, — сказала Делани. — Я разослала порядка 500 предложений заполнить веб-анкету. Кроме того, опросила по телефону 35 вероятных покупателей игры, подобной нашей. В итоге мы получили ответы от 75 потенциальных покупателей. Цель опроса заключалась в выяснении, какие из идентифицированных нами функций являются обязательными, т.е. такими, которые обязательно должны присутствовать в игре. Мы также

хотели узнать, какие функции являются привлекательными. Они необязательно должны иметься в игре, но их включение делает игру более привлекательной и позволяет повысить ее цену. Наконец, существуют такие функции, которые мы называем линейными. Чем больше линейных функций присутствует в продукте, тем больше люди платят за него. Примером линейных функций является мощность двигателя автомобиля, количество каналов спутникового телевидения и количество уровней сложности в игре. Не буду подробно рассказывать о том, как я проводила исследование. Но если вам интересно, могу назвать нужную главу в книге (см. главу 11 «Приоритизация желательности»). Результаты исследования я обобщила здесь. — Делани раздела присутствовавшим распечатку табл. 23.12.

Таблица 23.12. *Распределение результатов опроса пользователей*

История или тема	Привлекательная	Линейная	Обязательная	Безразличная	Обратная	Сомнительная	Категория
Слабый противник	8	34	50	4	4	0	Обязательная
Средний противник	12	34	51	1	1	1	Обязательная
Сильный противник	10	16	39	32	2	1	Безразличная Обязательная
Игра с другим человеком	5	23	14	58	1	0	Безразличная
Эстетически приятная	20	63	24	2	0	1	Линейная
Подсказки, предупреждение о неудачном ходе	47	21	21	0	0	1	Привлекательная
Интерактивное руководство	35	24	21	18	2	0	Привлекательная
Фоновая музыка	15	48	21	12	2	2	Линейная
Онлайновая помощь	10	42	36	5	3	4	Линейная
Мигание выигрышной фигуры	18	12	55	9	3	3	Обязательная
Статистика	12	28	26	34	0	0	Безразличная
Комментарии к сохраненным партиям	14	15	32	36	1	2	Обязательная Безразличная

— Делани, пунктов здесь намного меньше, чем наших историй, — сказал Прасад. — Почему?

— Чем длиннее анкета, тем реже ее заполняют, — ответила Делани. — Поэтому я объединила истории по своему усмотрению. Например, «Эстетически приятная» в этой таблице охватывает три наши истории: о красивой графике, о начальном экране и о возможности переключаться между деревянной и металлической доской и фишками. В то же время, если объединить слишком много историй в одном вопросе, полученные ответы могут оказаться не слишком полезными.

— Я могу связать первые три колонки с твоим описанием трех типов функций, Делани. Но что означают категории «Безразличная», «Обратная» и

«Сомнительная»? — спросила Роуз.

— Хороший вопрос. Безразличная означает, что у человека нет никаких заметных предпочтений в отношении функции, — объяснила Делани. — Форма представления вопросов позволяет нам не учитывать случайное мнение. Это, например, может произойти, когда кто-то говорит, что ему не нравится фоновая музыка, но ему также не по вкусу ее отсутствие. Такое случается. Люди, бывает, путают или неправильно понимают вопросы в спешке. Обратите внимание, что доля обратных и сомнительных функций довольно невелика.

— Делани, я правильно это понимаю? — не успокоилась Роуз. — Здесь написано, что слабый и средний противник обязательны. А в строке «Сильный противник» стоят сразу две категории — безразличная и обязательная. Что это означает? Почему ты не выбрала вариант, на который приходится наибольший процент ответов?

— Да, слабый и средний противники обязательны. Причина, по которой мы видим два пиковых значения у сильного противника, скорее всего, в том, что мы имеем дело с двумя разными аудиториями — с теми, кто уже играет в Navannah и поднаторел в игре, и теми, кто пока не успел сделать этого. Опытные игроки считают сильного противника обязательным, если купят игру. Новички говорят нам, что им достаточно и среднего противника.

— Делани, я заметил, что в списке не так много функций, которые наша аудитория считает привлекательными, — сказал Фрэнк, и это больше было похоже на вопрос, чем на утверждение.

— Это действительно так. Но небольшое число привлекательных функций не означает, что сам продукт не будет привлекательным, — ответила Делани. — Не забывайте, что привлекательными считаются функции, которых не ожидают. Если я забираюсь в новенький Porsche, то вряд ли найду там много действительно неожиданного. Однако сам автомобиль все равно хорош. Но мне кажется, я понимаю, что тебя беспокоит. Мы еще не идентифицировали все функции, которые необходимы этому продукту. В каждом продукте должна быть как минимум пара привлекательных функций — нечто такое, что заставляет пользователя сказать: «Вау!» Опрос, похоже, говорит, что мы пока не нашли ничего такого. Именно поэтому я прошу выделить мне время в первой итерации для продолжения исследования. Я хочу провести ряд открытых обсуждений с некоторыми участниками опроса. Надеюсь, это поможет найти одну-две привлекательные функции.

— Правильно, это было бы очень полезно, — сказал Фрэнк. — У кого-нибудь есть еще вопросы к Делани? — Фрэнк выдержал паузу, прежде чем продолжить: — Если нет, то давайте перейдем к обсуждению того, что нужно включить в релиз Navannah.

Вопросов больше не было. Команда начала заниматься планированием релиза с приоритизации функций.

— Есть истории, которые я считаю обязательными даже без учета результатов опроса, — сказала Делани и выложила на стол несколько карточек историй. (Карточки историй и оценки, которые она написала на них, представлены в табл. 23.13.)

Таблица 23.13. Истории, которые Делани считала очевидно обязательными и не включала в анкету

Текст истории	Оценка
Как игрок я могу начать новую игру	1
Как игрок я могу восстановить сохраненную игру	2
Как игрок я хочу, чтобы компьютер распознавал выигрышную фигуру	2
Как игрок я хочу ставить фишку на доску с помощью клавиатуры или мыши	3
Как игрок я хочу видеть визуальный индикатор очередности хода	2
Как игрок я хотел бы видеть визуальный индикатор последней поставленной фишки (возможно, сделать ее мигающей)	2
Как игрок я хотел бы иметь возможность отменять и повторять ходы	2
Как игрок я хотел бы иметь возможность сохранять партии	3
Как игрок я хочу иметь возможность закончить игру	1
Как игрок я хочу возобновлять игру так, чтобы можно было отказаться от текущей игры и начать новую	1
Как игрок я хотел бы, чтобы компьютеру требовалось на ход не более двух секунд на 2,0 ГГц РС	8
Как игрок я хотел бы просматривать ходы с тем, чтобы иметь возможность анализировать сыгранные партии	5
Всего	32

— Ни одна из этих историй не должна быть неожиданной, — продолжила Делани. — Мы видим эти функции в каждой разыгрываемой партии. Их поддержка — это стоимость входа.

Делани взяла еще одну группу карточек.

— На этой части стола я разложу истории, которые наши клиенты считают обязательными. — Эту группу карточек она разместила так, как показано в табл. 23.14.

Таблица 23.14. Обязательные истории, идентифицированные на основе первоначального исследования Делани

Текст истории	Оценка
Как игрок я могу выбирать уровень сложности	1
Как игрок я могу играть против слабого движка, распознающего только кольца	8
Как игрок я могу играть против слабого движка, распознающего только мосты	5
Как игрок я могу играть против слабого движка, распознающего только вилки	8
Как игрок я могу играть против движка среднего уровня	8
Как игрок я могу играть против движка высокого уровня	20
Как игрок я хотел бы иметь возможность использовать систему для игры с другим человеком на моем компьютере	3
Как игрок я хочу, чтобы все фишки выигрышной фигуры мигали или подсвечивались с тем, чтобы можно было видеть выигрышную фигуру	3
Всего	56

— Делани, а почему ты включаешь сильный движок сюда? — спросил Аллан. — Мне это нравится, но в соответствии с тем, что ты нам показывала несколько минут назад, эта функция считается одновременно и обязательной, и безразличной.

— Хорошее замечание, Аллан, — ответила Делани. — Я просто рекомендую считать эту функцию обязательной, поскольку, на мой взгляд, многие пользователи перерастут то, что мы определили как движок среднего уровня. Эти пользователи забросят игру, если мы не включим в нее движок высокого уровня.

— Окей.

Фрэнку очень нравился этот процесс. Раньше команда нередко отдавала все вопросы определения функциональности продукта ему как менеджеру по продукту и Делани как аналитику. А теперь он наслаждался тем, что вся команда участвовала в обсуждении и внимательно относилась к тому, о чем шла речь. Это не могло не пойти на пользу новому продукту.

— Еще одна история, которую я включила в категорию обязательных и которую стоит обсудить, это «Как игрок я хотел бы иметь возможность использовать систему для игры с другим человеком на моем компьютере». Я считаю ее обязательной, поскольку мы уже говорили, что она является хорошей контрольной точкой. Следующая группа историй, — сказала Делани, выкладывая очередную стопку карточек, — относится к категории линейных. Иначе говоря, чем их больше, тем лучше, но ни одна из них не является обязательной.

Делани разложила истории на столе так, как показано в табл. 23.15.

Таблица 23.15. Истории, которые в соответствии с исследованием Делани являются линейными (чем их больше, тем лучше)

Текст истории	Оценка
Как игрок я хочу видеть привлекательный начальный экран при загрузке игры	5
Как игрок я хочу видеть привлекательный фон вокруг игрового поля, сочетающийся с выбранной доской	5
Как игрок я хотел бы иметь возможность выбирать между деревянной и металлической доской и фишками	8
Как игрок я хочу, чтобы в игре был музыкальный фон	5
Как игрок я могу выбирать музыкальный фон игры	1
Как новый игрок я хочу иметь доступ к онлайн-справочной системе	8
Всего	32

— Художественное оформление нельзя считать чем-то опциональным, — сказала Роуз. — Нам обязательно нужны красивый начальный экран и приятные для глаза доски и фон.

— Я не говорила, что эти функции опциональны, Роуз, — возразила Делани. — Ты совершенно права — они обязательно нужны. Существует определенный базовый уровень, ниже которого опускаться нельзя, и игра должна быть эстетически приятной, именно так и звучал вопрос в анкете. Но такие функции линейны, поскольку чем их больше, тем лучше. Два варианта игровой доски лучше, чем один, а три — еще лучше.

— Это понятно, — сказал Фрэнк. — Но у тебя есть еще какие-то карточки. Что за истории на них?

— Эти карточки — привлекательные истории. Потенциальные покупатели не ожидают увидеть ни одну из них, но мы должны включить парочку таких функций в продукт по той причине, что они реально повышают удовольствие от игры. И, как я уже говорила, наличие подобных функций позволяет запрашивать более высокую цену (см. табл. 23.16).

Таблица 23.16. Привлекательные функции в соответствии с исследованием Делани

Текст истории	Оценка
Как игрок я хотел бы иметь возможность иногда пользоваться подсказками	1
Как новый игрок я хочу иметь возможность ознакомиться с интерактивным самоучителем по игре	8
Как новый игрок я хотел бы получать предупреждение о неудачном ходе и иметь возможность отменять его	8
Как новый игрок я хотел бы, когда наступает моя очередь делать ход, видеть клетку, на которую нужно поставить фишку, чтобы ее не занял компьютер и не обыграл меня	3
Всего	20

— После привлекательных функций, — продолжила Делани, — у нас остаются две истории, к которым потенциальные игроки безразличны. Я не думаю, что нам нужно включать их в релиз. — Она выложила на стол карточки (табл. 23.17). — Нам вряд ли удастся продать больше или продавать по более высокой цене в результате добавления функций, к которым игроки безразличны.

— Здорово! — сказал Фрэнк. — Это больше информации, чем у нас обычно бывает на данном этапе. Отличная работа, Делани. Все это получено в результате опроса?

— Да. Анкета специально была сделана несложной, в ней пользователей просто спрашивали о том, как они относятся к наличию той или иной функции и к ее отсутствию. Из-за того, что я объединила некоторые наши пользовательские истории в темы, в анкету вошли всего лишь 24 вопроса. При разговоре по телефону с учетом вступительной болтовни на нее уходило минут 15. Наш веб-сайт отслеживал время, затрачиваемое на заполнение формы. В среднем оно занимало семь минут. Одному посетителю потребовалось 40 минут, но я подозреваю, что он просто уходил или разговаривал с кем-то по телефону.

— Итак, Фрэнк, как ты расставишь приоритеты? — спросила Саша.

— Нам нужно реализовать все функции, которые Делани отнесла к обязательным или которые попали в категорию обязательных в результате опроса.

— По нашим оценкам, их суммарный размер составляет 88 пунктов, — сказала Саша.

— В отношении функций, которые Делани определила как линейные, я согласен с Роуз: нам необходимо хорошее художественное оформление, — продолжил Фрэнк. — Все эти истории должны войти в релиз. Кроме того, я слабо представляют себе игру без музыкального фона. Возможно, у пользователя не будет выбора, но эта история тянет всего на один пункт,

поэтому, на мой взгляд, она нужна. Ну и, конечно, онлайн-помощь. Не все покупатели этой игры будут знакомы с правилами.

Таблица 23.17. *Функции, которым пользователи были безразличны в соответствии с исследованием Делани*

Текст истории	Оценка
Как игрок я хочу, чтобы система отслеживала, сколько партий я выиграл, а сколько проиграл (в зависимости от уровня сложности?)	3
Как игрок я хотел бы добавлять комментарии к сохраненным партиям	3
Всего	6

— Таким образом, ты хочешь получить все линейные функции. Это еще 32 пункта, — сказала Саша.

— Я знаю. Я, как и любой другой менеджер по продукту, хочу видеть в релизе все эти функции.

— Я не возражаю, а просто хочу понять, какой продукт ты желаешь получить через четыре месяца. Суммарный размер историй на данный момент составляет 120 пунктов.

— Ты думаешь, на их реализацию может потребоваться больше четырех месяцев? — спросил Фрэнк с беспокойством.

— Пока что не знаю. Давайте определимся с тем, чего мы хотим, а потом посмотрим, сколько времени на это потребуется, — ответила Саша.

— Резонно, — заметил Фрэнк. — Что до привлекательных функций, то, я думаю, здесь нам нужны подсказки, ну и, возможно, самоучитель. Не исключено, что нам придется отказаться от онлайн-помощи, но мне хотелось бы включить в план и то и другое. Еще мне нравится идея предупреждения о неудачном ходе.

— Не забывайте, что эта история оценивается в восемь пунктов, — напомнил Аллан. — Я должен сделать так, чтобы движок мог определять хорошие ходы. Можно, конечно, расширить его возможности до идентификации неудачных ходов, но это не так просто.

— Именно это меня и беспокоит. Давайте пока обойдемся без этой функции, но все же включим историю, связанную с предупреждением игрока об угрозе проигрыша, если он не закроет определенную клетку и не заблокирует ход компьютера.

— Это 12 пунктов из списка привлекательных функций. Будем ли включать что-то из категории безразличных функций? — спросила Саша.

— Не думаю, — ответил Фрэнк. — Что у нас получается в сумме?

— Ты выбрал 88 пунктов из категории обязательных функций, 32 пункта из категории линейных и 12 из категории привлекательных. Всего получилось 132 пункта, — сказала Саша.

— И вы реализуете это за две итерации, так? — спросил Фрэнк с улыбкой.

— Если сможем, то да.

— А если серьезно, то как определить, сколько времени на это потребуется?

— Лучше всего было бы выполнить три итерации и посмотреть, — объяснил Карлос. — После каждой итерации можно суммировать количество пунктов для реализованных историй. Мы называем это скоростью. Скорость варьирует от итерации к итерации, но в среднем остается довольно стабильной.

— Да, — оживился Аллан, — наши оценки могут быть ошибочными. К тому же история может оказаться удачной или неудачной.

— После трех итераций мы должны получить надежный показатель темпов прогресса команды. Тогда суммарный объем оставшейся работы можно разделить на скорость и получить ожидаемое время завершения, — сказал Карлос.

— И такая оценка будет довольно точной? — спросил Фрэнк.

— Должна быть. После трех итераций у вас будет хорошее представление о том, сколько пунктов вы можете реализовать за двухнедельную итерацию, — ответил Карлос.

— Не следует забывать, что я собираюсь поговорить еще с некоторыми потенциальными покупателями и могу обнаружить новые привлекательные функции, — сказала Делани. — Как уже говорилось, наши существующие привлекательные функции слабоваты, и было бы неплохо дополнить их.

— Но можно хоть как-то оценить сроки прямо сейчас?

— Конечно, Фрэнк, — сказал Карлос. — Вы планировали реализовать четыре истории в первой итерации. Мы можем сложить пункты, связанные с каждой из них, и назвать это плановой скоростью. Никто не знает, насколько она реальна, но это наше первоначальное предположение.

— Я уже сделал это, — сказал Прасад. — Сумма равна 18. Восемь пунктов приходится на историю «Как игрок я могу играть против слабого движка, распознающего только кольца», пять пунктов — на историю «Как игрок я могу играть против слабого движка, распознающего только мосты», два пункта — на историю «Как игрок я хочу, чтобы компьютер распознавал выигрышную фигуру» и три пункта — на историю «Как игрок я хотел бы иметь возможность использовать систему для игры с другим человеком на моем компьютере».

— Сколько итераций потребует реализация проекта при скорости 18?

— Чуть больше семи итераций. У нас 132 пункта. Семь на 18 — это 126. Итого шесть пунктов сверх семи итераций, — сказал Прасад.

— Можем ли мы сказать, что нам нужно семь итераций? — спросил Фрэнк.

— Можем, но делать этого не стоит, — ответил Карлос. — Я предпочитаю давать оценки в виде диапазона. Если посмотреть на объем работы, которая приходится на восьмую итерацию, то лучше всего сказать, что проект займет от шести до 10 итераций.

— Мне как-то не хочется, чтобы этот проект растягивался на 20 недель. Но ты говоришь, что он может занять и 12 недель, так ведь? — спросил Фрэнк.

— Может, но с равным успехом может растянуться до 20, — настаивал Карлос. — Чтобы этого не случилось, ты можешь удалить наименее приоритетные функции, если скорость покажет, что мы ближе к 10 итерациям. Не забывайте также, что Делани активно занимается поиском новых функций. Если она обнаружит что-то, нам придется отодвинуть срок или отказаться от каких-нибудь текущих функций.

— Ну хорошо, я смогу объяснить руководству разброс от 12 до 20 недель, — сказал Фрэнк. — Давайте теперь займемся нашим начальным планом.

— Фрэнк, я опять повторяю, что лучше всего подождать *как минимум* две недели, дать команде возможность выполнить итерацию, а потом построить начальный план на этом единичном наблюдении фактической скорости, — возразил Карлос.

— Понимаю. Но все хотят знать хотя бы на уровне идеи, как выглядят сроки реализации этого проекта.

— Объясни им, что мы можем предоставлять более точную информацию каждые две недели. После первых трех итераций мы будем знать намного больше, — сказал Прасад.

— Так нам нужен план того, над чем мы будем работать в каждой итерации? — спросил Фрэнк.

— Не совсем, — ответил Карлос. — Самое важное — это знать, что будет разрабатываться в следующей итерации. Мы делали это, когда планировали итерацию. Следующий важный момент — определить те функции в конце списка, которые можно включать в релиз, а можно и не включать.

— Я понимаю, зачем нужно знать главные приоритеты, но зачем нам функции из нижней части списка? — заметил Фрэнк.

— Это не так важно для первого совещания, но ситуация меняется по мере приближения к завершению релиза. Например, вряд ли мы захотим, чтобы маркетинговая служба разработала упаковку с надписью «Теперь с музыкальным фоном», если эта функция может быть выброшена в последнюю минуту.

— Хорошо, но почему бы не потратить сегодня немного времени на планирование того, что мы будем делать в каждой итерации?

— Нет смысла, — ответил Карлос. — Через две недели мы будем знать намного больше, чем сегодня. Зачем планировать то, что мы будем делать в итерации, до того, как возникнет необходимость? Этого не нужно делать. В

крупном проекте, особенно с участием нескольких команд, работу которых необходимо координировать, можно забегать вперед на две-три итерации. Но в нашем проекте это ни к чему.

— Так мы закончили с планированием релиза? — спросил Фрэнк.

— Да. Я сейчас соберу карточки историй, которые вы определили как действительно необходимые в релизе, это и будет нашим планом, — сказал Карлос. — Прежде чем мы вновь соберемся через две недели, вы с Делани должны рассмотреть истории и определить те, которые, вероятнее всего, войдут в следующую итерацию. На совещании мы поговорим о том, почему вы остановились именно на этих историях. Другие участники могут попросить вас пересмотреть или добавить одну или две истории, которые они считают рискованными или о которых получили новую информацию в процессе разработки. Так или иначе, окончательное решение о приоритизации принимаете вы.

— Не пойми меня неправильно, — сказал Фрэнк, — но я не уверен, что справлюсь с этим сегодня.

— Естественно, — сказал Карлос, — однако выбор историй, которые нужно взять в работу в следующей итерации, это задача всей команды. Если программисты начинают выбирать технические прикаты или функции, которые невозможно увидеть, я останавливаю это и предлагаю им работать над другими историями. Я полагаю, ты доволен историями, которые были выбраны?

— Абсолютно. Будет чудесно увидеть все это через две недели.

Две недели спустя

— Жаль, что нам не удалось реализовать все. Тем не менее меня удивляет, как много мы сделали, — говорила Саша коллегам, пока они ждали начала совещания.

— К сожалению, нам пришлось отказаться от истории «Как игрок я хочу, чтобы компьютер распознавал выигрышную фигуру», но ты мне здорово помогла, Саша. Спасибо, — благодарил ее Аллан.

— Не стоит, Аллан. Все жалеют об этом, но продемонстрировать движок, который умеет выстраивать кольца и мосты, было важнее, — ответила Саша.

— Все готовы к представлению демоверсии? — спросил Фрэнк, который вошел в зал точно в 9:00. Он поставил на стол коробку с булочками.

— Конечно, — ответила Саша. — На анализе результатов итерации будет присутствовать кто-нибудь из руководства?

— Обещали быть Фил и Лора, — сказал Фрэнк.

— А еще придет пара инженеров из других команд. Они очень интересуются agile-подходом к разработке, — добавил Аллан.

В течение следующего получаса команда показывала свои достижения. Она продемонстрировала игровой движок, выстраивающий кольца и мосты. Помимо этого, она показала, как два человека играют друг с другом, делая ходы с помощью общей клавиатуры. Команда даже связала две функции, чтобы человек мог играть против компьютера. Человек все время выигрывал, что было неудивительно, учитывая ограниченные возможности генератора ходов.

Графика была рудиментарной — только наброски Роуз. Однако Фил, генеральный директор, пришел в восторг, увидев, что всего через две недели пусть небольшая, но реальная часть игры была реализована и функционировала. Еще больше ему понравилось то, что Прасад протянул клавиатуру и предложил ему сыграть самому.

— Это фантастика, — заявил Фил, поднимаясь с кресла. — Так вы будете показывать что-то новое каждые две недели?

— Да. Каждые две недели здесь в 9:00, — ответила Саша.

— Я обязательно приду. Я включу эти совещания в свой график. Эта получасовая демонстрация работы реальной программы более полезна, чем отчет о ходе работ, который я обычно читаю. А вы определились со сроками? Могу я сейчас об этом спросить?

— Конечно, — ответил Фрэнк. — Сейчас мы придерживаемся нашей прежней оценки: 12–20 недель. После этого совещания мы будем планировать следующую итерацию. Через две–четыре недели мы будем знать больше. А пока нам необходимо определить сроки, которые можно сообщить издателям.

— Отлично, — сказал Фил.

Он и Лора вышли из конференц-зала. За ними последовали еще человек пять, которые приходили познакомиться с проектом.

Планирование второй итерации

— Если я не ошибаюсь, — сказал Фрэнк, — наша плановая скорость составляла 18 пунктов.

— Именно столько, — ответила Делани.

Однако нам не удалось завершить историю о распознавании выигрышной фигуры, поэтому наша скорость оказалась меньше 18. Можем ли мы дать себе один пункт за выполнение половины этой истории? — спросил Фрэнк.

— Нет, — ответил Карлос. — Обычно к пунктам применяют правило «всё

или ничего». Очень трудно определить, какую часть пунктов можно себе присвоить, поскольку на деле мы не знаем объема оставшейся работы. Вместо гаданий мы просто не присваиваем пункты до тех пор, пока история не будет реализована целиком.

— Поэтому наша скорость составила 16 с учетом того, что мы оценили незаконченную историю в два пункта, — сказала Роуз.

— Так мы должны исходить из того, что выполнение проекта займет больше времени. Правильно? — спросил Фрэнк.

— Мы все еще не вышли из диапазона 12–20 недель, который представили руководству, — сказала Саша. — Мы начали, имея 132 пункта, выполнили 16, значит, у нас осталось 116 пунктов. Делим 116 на 16 и получаем 7,25. Для страховки скажем, что это скорее восемь двухнедельных итераций. С учетом прошедших двух недель в сумме получается 18 недель.

— Это на две недели больше, чем четыре месяца, Фрэнк. Это существенно? — спросил Прасад.

— Нет, на самом деле. Релиз не привязан к конкретной дате. Мы просто хотим сделать его как можно быстрее, — ответил Фрэнк. — Да, должен вам сказать, мы не закрыли договор на Deer Black & White с японским издателем. Они все еще хотят издавать игру. До подписания окончательного договора осталось совсем немного. Однако они не хотят начинать продажу игры, пока не запустят серьезную маркетинговую программу с рекламой во всех крупнейших журналах. На это может уйти четыре месяца. Деньги же мы начнем получать только через квартал после начала продаж. Таким образом, прежде чем Deer Black & White принесет нам какие-нибудь деньги, может пройти восемь месяцев. Было бы хорошо, если бы Navannah дала быструю отдачу. Пусть даже не в таком масштабе, как Deer Black & White. Но из-за того, что она не требует предварительной рекламной раскрутки, выручка от нее может поступить быстрее.

— Фрэнк, прошу прощения, но из-за более значительного объема тестирования, которым мне пришлось заниматься, я мало что сделала для исследования продукта во время этой итерации, — сказала Делани.

— Ничего страшного, но пусть это будет приоритетом в следующей итерации, даже в ущерб объему тестирования, — успокоил ее Фрэнк.

— Итак, Фрэнк, что бы ты хотел видеть реализованным в следующей итерации?

Команда спланировала вторую итерацию точно так же, как она сделала это с первой. Когда обсуждение закончилось, команда обязалась реализовать пользовательские истории, представленные в табл. 23.18.

Таблица 23.18. Пользовательские истории, включенные в план второй итерации

Текст истории	Оценка
Как игрок я хочу, чтобы компьютер распознавал выигрышную фигуру	2
Как игрок я могу играть против слабого движка, распознающего только кольца	8
Как игрок я хотел бы иметь возможность выбирать между деревянной и металлической доской и фишками	8
Всего	18

— Но, если наша скорость в прошлой итерации была равна 16, почему мы в этот раз включаем в план 18 пунктов? — спросил Фрэнк.

— Мы используем скорость в качестве ориентира для оценки прогресса по отношению к плану релиза, — сказал Карлос. — Не нужно подгонять каждую новую итерацию к скорости в предыдущей итерации или к средней скорости в нескольких последних итерациях. Планирование итерации осуществляется исходя из того, какой объем работы мы можем выполнить. Пункты добавляются только после того, как мы решили, что можем взять обязательство по их реализации. Когда мы делаем так, скорость должна быть близкой к исторически среднему значению, но она неизбежно будет варьировать.

— Почему?

— По двум основным причинам, — начала Саша. — Во-первых, в этот раз мы будем более эффективно использовать Роуз. Она прежде всего художник, а не тестировщик. Здорово, что она помогла нам с тестированием в прошлый раз, но в этой итерации ей нужно будет заняться художественным оформлением деревянных и металлических досок и фишек. Я займусь кодированием, чтобы пользователь мог выбирать между ними, а Прасад, скорее всего, один справится с тестированием этой истории. Кроме этого, основная задача по реализации данной восьмипунктовой истории лежит на Роуз. Во-вторых, мы получили некоторый задел с историей по распознаванию выигрышной фигуры. Работа над ней началась в прошлой итерации, но не завершилась. Я довольно оптимистично смотрю на выбранные истории. Несмотря на те же 18 пунктов, в них получается на 15 часов меньше, чем в предыдущей итерации.

— Убедили, — сказал Фрэнк. — Давайте так и сделаем.

Две недели спустя

При анализе следующей итерации команда вновь продемонстрировала хороший прогресс. Она реализовала все запланированные истории. А у

Делани появилось время для завершения исследования продукта. На совещании вновь присутствовали Фил и Лора. Поскольку новость о быстрой разработке уже разлетелась по компании, на этот раз на обсуждение пришло больше представителей других команд.

— Игра начинает приобретать очертания. Не могу поверить, что ваш генератор ходов уже может обыграть меня, — сказал Фил. — Роуз, эти первые две доски выглядят фантастически. Сроки у нас все еще прежние? Через 8–16 недель с текущего момента?

— Темп нашего продвижения именно такой, как мы ожидали, — сказал Фрэнк. — Поэтому мы укладываемся. Делани провела дополнительное исследование продукта в течение этой итерации и обнаружила несколько интересных функций, которые мы, возможно, включим в продукт. Я дал добро на это, но нужно еще узнать мнение команды. Я могу допустить изменение сроков. Мы считаем, что новые функции принесут нам больше денег. Теперь мне нужно взвесить дополнительные затраты на разработку и дополнительное время.

— Мне нравится то, что ты сказал насчет увеличения выручки, — сказала Лора, финансовый директор.

— Мне тоже, — поддержал ее Фил. — Но я не в восторге от возможного переноса сроков.

— Речь идет не о переносе сроков, Фил, — сказал Фрэнк. — Если имеются в виду первоначально запланированные 12–20 недель, то мы не видим здесь проблемы. Но после этого совещания мы собираемся обсудить, насколько больше времени займет создание более ценной игры. А после мы посмотрим, нет ли таких функций, от которых можно отказаться, чтобы уложиться с более ценной игрой в те же 20 недель.

— Я всегда за более ценные игры. Сообщите мне о своем решении, — сказал Фил, покидая конференц-зал. Вслед за ним вышли и остальные гости. В зале осталась только проектная команда.

Пересмотр плана релиза

— Фрэнк уже сообщил вам, что я нашла несколько новых привлекательных функций, — сказала Делани. — Кое-кто из вас слышал, как я обсуждала их по телефону с потенциальными покупателями. Хорошая новость заключается в том, что наша целевая аудитория действительно хочет получить две функции, о которых мы не подумали. Во-первых, покупатели хотят, чтобы компьютерный игрок имел несколько характеров, разные стили игры. Один может быть очень агрессивным, другой — консервативным. Многие хотели бы, чтобы компьютер поддразнивал

соперника. Он должен говорить колкости после плохого хода, смеяться, когда соперник проигрывает... ну и т.п.

— Звучит занятно, — вмешался Аллан. — Это изменит генератор ходов, но, пожалуй, не сильно. Его уже сейчас можно настроить на более агрессивную или более консервативную игру.

— Это первая новая функция, — продолжила Делани. — Вторая — это онлайн-игра. Люди хотят играть друг с другом в сети.

— Это огромная задача, — сказала Саша. — Нам нужно дополнительное место в центре обработки данных, надо купить дополнительное оборудование и нанять обслуживающий персонал.

— Мы можем с кем-нибудь скооперироваться для этого, — заметил Фрэнк. — Это не такой уж сложный вопрос, но, ты права, он большой. Делани, ты представляешь, какая функция более важна?

— Характеры. Хорошие характеры должны стать действительно очень привлекательными. Это функция, которая реально понравится людям, но которую они совершенно не ожидают, — ответила Делани.

— Делани, ты только что сообщила, что объем работ поползет. Но мы ведь и так уже у верхнего края нашего 12–20-недельного диапазона, — заметил Прасад. — Почему же ты говоришь, что это «хорошая новость»? У тебя есть еще что-то для нас?

— Нет, это все, Прасад, — ответила Делани. — Но это хорошая новость. Мы нашли две новые функции, которые помогут нам продать больше игр и по более высокой цене. Конечно, если мы реализуем эти функции, сроки изменятся. Но это хорошая проблема. Мы можем уложиться в текущие сроки и выручить за продукт определенные деньги. Или мы можем изменить сроки и рассчитывать на более существенные деньги. Планы — это всегда предсказания, привязанные ко времени. Мы просто нашли более приятную перспективу, на которую можно нацеливаться. Я уже написала пользовательские истории для обеих функций. Нам нужно оценить их и решить, хотим ли мы изменить план так, чтобы включить в него и ту и другую.

— Давайте займемся этим, — сказал Фрэнк. — Я полностью готов изменить сроки проекта, если мы сможем больше заработать. И это не такая уж большая отсрочка. Эти функции выглядят очень заманчиво.

— Но, Делани, почему анкета, которую ты рассылала, не позволила идентифицировать эти функции? — спросил Прасад.

— Анкеты хороши для приоритизации того, что мы уже знаем, — ответила Делани. Однако они неэффективны для выявления функций, о которых мы не подозреваем. В некоторых анкетах, которые мы получили, упоминаются характеры и онлайн-игра, но не так часто, чтобы я могла увидеть тенденцию. В течение последней итерации я переговорила с наиболее вероятными покупателями, которые подтвердили

перспективность этих идей, а потом скорректировала финансовые модели, чтобы понять, какую дополнительную выручку могут принести новые функции.

— Мне нравится эта мысль, — сказал Аллан. — Если игра принесет больше денег, возможно, Фрэнк пригласит нас в какое-нибудь более дорогое местечко, чем Fatburger, чтобы отпраздновать выпуск релиза.

— Мне понравились бургеры, когда мы отмечали выпуск Deer Black & White, — сказал Прасад.

— А мне нет. После нынешнего релиза, я думаю, Фрэнк раскошелится на суши, — заметила Роуз.

Команда оценила новые истории, записанные Делани. При этом одни первоначальные истории были разбиты, а другие объединены. Когда обсуждение закончилось, истории, связанные с онлайн-игрой, оценили в 30 пунктов. Созданию первого характера присвоили 15 пунктов, а каждому последующему — пять пунктов. Поразмыслив, команда пришла к выводу, что пяти характеров будет вполне достаточно. Это означало, что разработка характеров в сумме вносит 35 дополнительных пунктов.

Поскольку финансовые прогнозы Делани показывали, что характеры должны принести более значительную выручку, чем онлайн-игра, все согласилось, что в план следует включить именно их, а не онлайн-версию.

— В первых двух итерациях мы реализовали 16 и 18 пунктов. Теперь мы добавили 35 пунктов и, таким образом, вернулись туда, где были! — пошутил Аллан.

— Ты прав, Аллан, — сказал Карлос, доставая калькулятор. — Мы получили 133 пункта. При текущей средней скорости 17 пунктов на итерацию нам нужно 7,8 итераций.

— Восемь итераций плюс две, которые мы уже выполнили, итого 10 итераций, — сказал Фрэнк.

— Мы подошли ужасно близко к пределу, Фрэнк, — сказала Саша. — Да, похоже, мы справимся с работой за 10 итераций. Как хорошо, что мы установили первоначальный срок в виде диапазона и не сказали просто «восемь итераций». Все же если что-то пойдет не так — если история окажется больше, чем мы думали, или мы замедлимся, или кто-нибудь заболеет, — то нам может потребоваться больше восьми итераций.

— Не хочешь ли ты сказать, что нам нужно изменить сроки? — спросил Фрэнк.

— Не обязательно, — вмешался Карлос. — Если мы хотим остановиться на 10 итерациях, нужно посмотреть, нельзя ли будет отказаться от каких-нибудь низкоприоритетных историй в случае цейтнота. Если вы не готовы пойти на это, то нам нужно довести до всех, что проект может занять более 10 итераций.

— Пожалуй, при необходимости я могу отказаться от некоторых историй, — сказал Фрэнк.

— Хорошо. Помните, как я говорил вам о том, что полезно знать не только истории с высшим приоритетом, но и истории с низшим приоритетом? Пришло время взглянуть на нижнюю часть списка. На мой взгляд, чтобы сохранить ожидания на уровне 10 итераций, нужно идентифицировать истории, без которых можно обойтись, как минимум на 17 пунктов, — сказал Карлос.

— Вот парочка тех, без которых я могу прожить: «Как новый игрок я хотел бы получать предупреждение о неудачном ходе и иметь возможность отменять его» и «Как новый игрок я хочу иметь возможность ознакомиться с интерактивным самоучителем по игре». Каждая из них тянет на восемь пунктов. Еще, как я уже говорил, я вполне могу обойтись без историй о музыкальном фоне на шесть пунктов.

— Таким образом, остановиться нужно на этих, последних, историях. Если вы начнете выбиваться из графика к десятой итерации, выбрасывайте одну из них или сразу несколько, — сказал Карлос.

— А если мы уложимся в график и решим отказаться от реализации этих историй, то сможем выпустить релиз как минимум на одну итерацию раньше, так? Когда мы должны выпустить релиз, если откажемся от этих историй? — спросил Фрэнк.

— У нас включено в план 133 пункта. Вы готовы выбросить 22 пункта. Остается 111 пунктов, — сказал Карлос, отрываясь от калькулятора. — При скорости 17 вам нужно еще семь итераций. Это означает, что проект можно завершить за 18, а не за 20 недель.

— Пусть эти истории пока остаются в плане. Я сниму их при необходимости, но на них, пожалуй, стоит потратить две недели, — сказал Фрэнк. — Я хочу показать этот новый план Филу, он хотел узнать, что мы решили. Кто-нибудь хочет составить мне компанию?

Сопровождать Фрэнка вызвались Делани и Аллан. В случае Делани удивляться здесь нечему — как аналитик, она часто участвовала в принятии решений по продукту и срокам. А вот интерес Аллана удивил Фрэнка. Обычно Аллана, одного из самых консервативных технарей в Bomb Shelter, совершенно не трогали споры в сфере бизнеса. Однако с начала этого проекта четыре недели назад он по-настоящему стал участвовать в них. Фрэнк оценил перемены. Это позволяло принимать более качественные решения по продукту.

— Я иду прямо сейчас, — сказал Фрэнк. — Давайте покажем, что получилось.

— Может, лучше сначала подготовить демонстрационные материалы для Фила? — спросил Аллан.

— Лишнее. У него есть магнитно-маркерная доска. Мы нарисуем все на

ней.

Презентация пересмотренного плана у Фила

— Привет, Фил. Ты по-прежнему хочешь узнать, что мы решили по этим новым функциям в Navannah? — спросил Фрэнк.

— Конечно. Это те функции, о которых Делани говорила утром?

— Да.

— У меня есть примерно 15 минут. Хватит?

— С лихвой, — сказал Фрэнк. Затем он попросил Делани объяснить, как ее исследование привело к появлению новых идей по онлайн-игре и характерам.

— Фил, помнишь, как я несколько недель назад рассказывала об оценках историй в пунктах?

— Конечно.

— И о том, что пункты — это показатель размера. Мы начали со 132 пунктов. В первой итерации было реализовано 16 пунктов, во второй — 18. Аллан, можешь изобразить это в виде графика? — попросил Фрэнк, указывая на свободное место на доске Фила. Аллан подошел к доске и начертил то, что изображено на рис. 23.5. — То, что ты здесь видишь, — продолжил Фрэнк, — мы называем диаграммой выгорания релиза. Глядя на нее, ты можешь видеть наш темп прогресса. После двух итераций мы реализовали 34 пункта. Если продолжить линию, то ты увидишь, что мы завершаем работу на восьмой итерации. Это в сумме 16 недель.

Пока Фрэнк говорил, Аллан продлил линию, как показано на рис. 23.6.

— Команда Navannah сошлась на том, что добавление характеров, идентифицированное Делани, это хорошая идея.

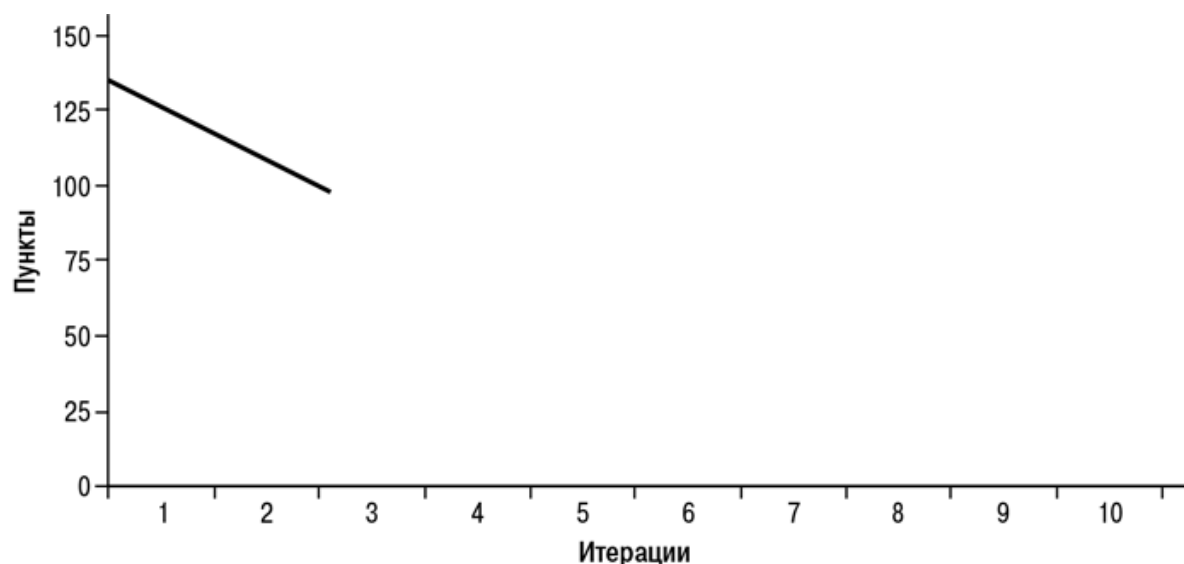


Рис. 23.5. Выгорание релиза Navannah после первых двух итераций

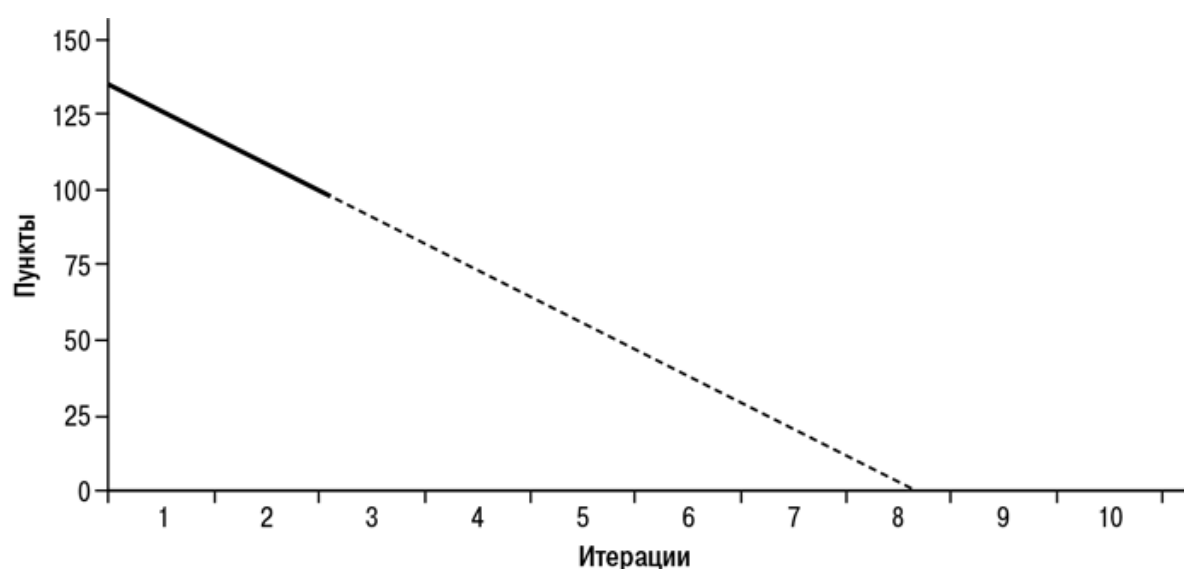


Рис. 23.6. Линия, построенная на основе выгорания в первых двух итерациях

— Сколько времени это займет? — спросил Фил.

— Мы считаем, что нам нужно пять характеров. Всего это 35 пунктов, — ответил Фрэнк.

— Это означает, что наше выгорание фактически возвращается сюда, — сказал Аллан. Он провел вертикальную линию вверх до уровня 133 пункта, как показано на рис. 23.7.

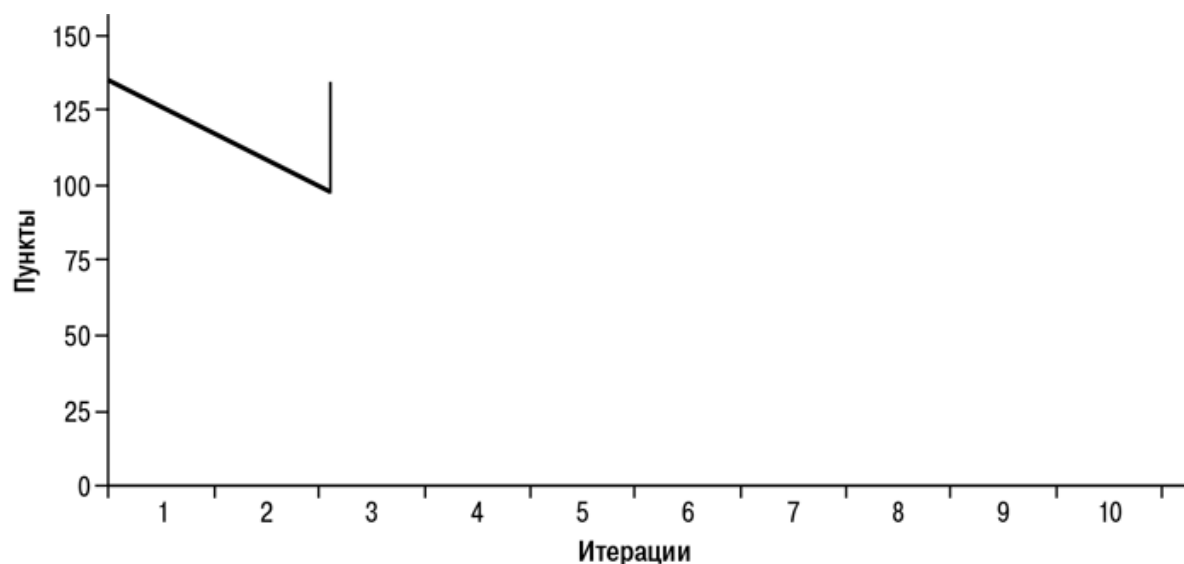


Рис. 23.7. Диаграмма выгорания релиза Havannah с учетом включения дополнительной функции

— М-м-м... Выглядит не очень хорошо. После двух итераций вы на том же месте, откуда начинали.

— В действительности наше положение намного лучше. За четыре недели мы заметно продвинулись вперед, — сказал Аллан, указывая на линию на диаграмме. — Мы устранили некоторые из наиболее рискованных частей проекта, было показано, как идет разработка генератора ходов. Хотя был сделан только слабый движок, я понял, как нужно подходить к созданию сильного движка. Мы многому научились. Мы нашли новую функцию, которую Делани определила как привлекательную для целевой аудитории. Обычно это выясняется намного позже. Если смотреть на то, где мы находимся, то действительно кажется, что у нас столько же работы, сколько было четыре недели назад, но сейчас я намного больше уверен в плане, чем тогда. И каждый раз, когда мы добавляем новую точку на этой диаграмме выгорания, моя уверенность крепнет, хотя далеко не все новости хорошие. Главная новость в том, что я могу быть уверенным в нашем прогрессе. — Аллан остановился. — Нет, наше положение намного лучше, чем четыре недели назад.

— Так сколько займет проект, если мы добавим характеры? — спросил Фил. К приятному удивлению Фрэнка, этот вопрос был адресован Аллану.

— Возможно, еще восемь итераций, или всего 20 недель. За итерацию мы выполняем работу объемом 17 пунктов. Это называется скоростью. Если продлить линию, показывающую скорость 17 пунктов на итерацию, то, как видно, мы завершим проект через восемь итераций. — Аллан продолжил линию, как показано на рис. 23.8. — Это очень близко к предельному сроку, поэтому мы идентифицировали несколько низкоприоритетных элементов, от которых можно отказаться, если начнем отставать или если решим

выпустить релиз через 18 недель вместо 20.

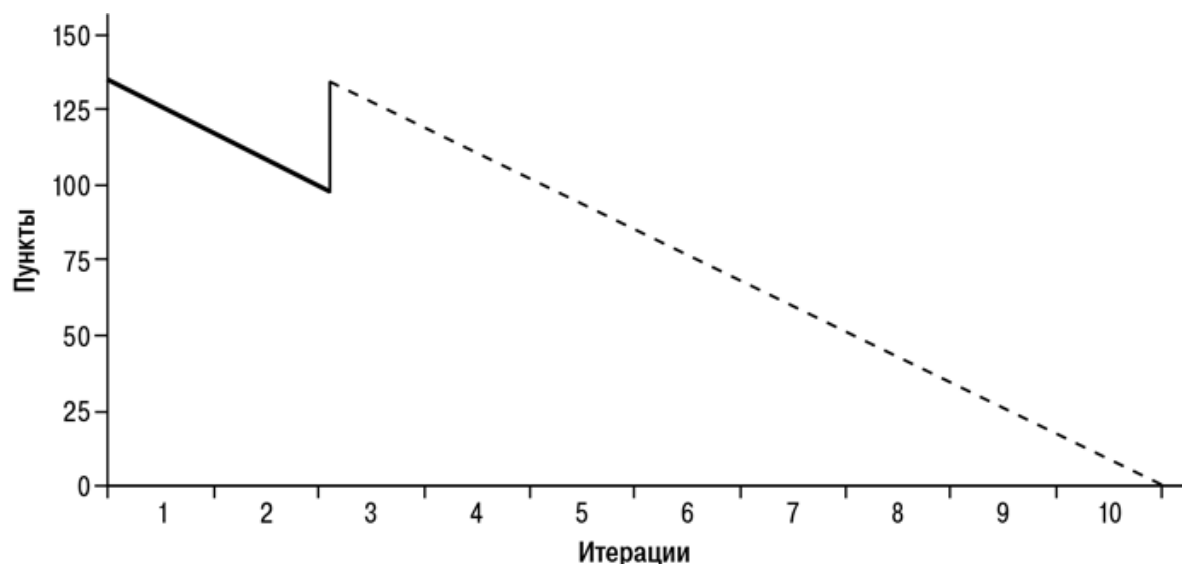


Рис. 23.8. Обновленная диаграмма выгорания, показывающая предсказанное завершение проекта Havannah

— И игра будет продаваться лучше или по более высокой цене? — спросил Фил.

— Да, я уверен в этом, — сказал Фрэнк. — Я хочу провести еще несколько исследований и построить модель, чтобы решить, следует ли нам продавать по текущей цене или можно увеличить ее на \$5.

— Вы меня убедили. Спасибо за презентацию, — сказал Фил.

Аллан начал стирать нарисованную диаграмму.

— Не надо. Пусть останется, — сказал Фил. — Я хочу обновлять ее каждые две недели.

— Спасибо, что уделил нам время, Фил, — сказал Фрэнк, когда они с Делани и Алланом покидали кабинет.

Восемнадцать недель спустя

— Отличная работа, ребята. Игра выглядит потрясающе, — сказал Фрэнк в конце совещания по анализу последней итерации. — Я не вижу недостатков. Движки функционируют очень хорошо. Роуз, я просто влюбился в твои характеры. Связь каждого из них с движком великолепна. Игроки выбирают характер игры, а не уровень сложности. Признаюсь, поначалу я скептически смотрел на мультяшные персонажи. Но мне нравится, как пираты подзадоривают игрока и как дружелюбная русалка ведет тебя к слабому движку. Ее диалог определенно будет помогать новым игрокам освоить игру. Думаю, мы готовы к отгрузке.

— Допускаю, что получилась отличная игра, но мне не нравится ощущение запаздывания. Первоначально мы думали, что ее можно сделать всего за 14 недель, но всем говорили, что за 12–20 недель. А в результате мы сдали работу еще на две недели позже, — сказал Прасад.

— Да, мы сделали ее за 22 недели, — сказала Саша. — Но мы и не говорили, что сдадим ее через 14 недель. После планирования первой итерации мы утверждали, что нам требуется от 12 до 20 недель, а более точная оценка появится только через шесть недель.

— К тому же, Прасад, не забывай, что мы могли бы закончить работу две недели назад, если бы захотели, — заметил Фрэнк. — Но мы решили внести дополнительные изменения в характеры по результатам, полученным от тестировщиков бета-версии. Им понравились характеры, но они дали нам несколько хороших рекомендаций по их улучшению. Мы заставили пиратов чаще отпускать замечания и сделали рэпера более разговорчивым. Конечно, первоначальный график был неправильным, но это же можно сказать и о дизайне продукта. Та игра не принесла бы нам такой финансовой отдачи, как эта. Вы играли в новую версию. Вы знаете, насколько лучше она сейчас, с запозданием всего на две недели по сравнению с первоначальной оценкой. Если бы мы выпустили версию 1.0 без характеров, а потом добавили их, я бы принял это. Но наша отрасль — вряд ли. Я очень доволен продуктом. Это больше, чем ожидалось, и всего на две недели позже, чем по оценке в начале проекта.

— А еще не забывайте, что Deer Black & White вышла позже на полгода, а не на две недели.

— Так что, Фрэнк, ты опять будешь угощать нас бургерами в честь этого релиза?

— Нет, на этот раз я куплю вам все, что вы пожелаете.

Список литературы

- Abdel-Hamid, Tarek K. 1993. Adapting, Correcting, and Perfecting Software Estimates: A Maintenance Metaphor. *IEEE Computer* 26 (3): 20–29.
- Ambler, Scott W. 2004. Less Is More. *Software Development*, November.
- Anderson, David. 2004. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. Prentice Hall.
- Andrea, Jennitta. 2003. An Agile Request For Proposal (RFP) Process. *Agile Development Conference*.
- Armour, Phillip. 2002. Ten Unmyths of Project Estimation. *Communications of the ACM* 45, no. 11: 15–18.
- Baker, Simon. 2004. Email to agileplanning@yahooogroups.com on November 3, 2004.
- Ballard, Glenn, and Gregory A. Howell. 2003. An Update on Last Planner. *Proceedings of the 11th Annual Conference of the International Group for Lean Construction*, pp. 1–13.
- Beck, Kent. 2002. *Test Driven Development: By Example*. Addison-Wesley.
- Beck, Kent, and Martin Fowler. 2000. *Planning Extreme Programming*. Addison-Wesley.
- Beck, Kent, et al. 2001. Manifesto for Agile Software Development, www.agilemanifesto.org.
- Bentley, Jon. 1999. The Back of the Envelope. *IEEE Software* 16 (5): 121–125.
- Bills, Mark. 2004a. ROI: Bad Practice, Poor Results. *Cutter IT Journal* 17 (8): 10–15.
- Bills, Mark. 2004b. Useful Back-of-the-Envelope Calculations. *Cutter IT E-Mail Advisor*, October 27.
- Boehm, Barry. 1981. *Software Engineering Economics*. Prentice Hall.
- Boehm, Barry. 2002. Get Ready for Agile Methods, with Care. *IEEE Computer*, 35 (1): 64–69.
- Boehm, Barry, and Richard E. Fairley. 2000. Software Estimation Perspectives. *IEEE Software* 17 (6): 22–26.
- Bossi, Piergiuliano. 2003. Using Actual Time: Learning How to Estimate. In *Extreme Programming and Agile Processes in Software Engineering: 4th International Conference*, edited by Michele Marchesi and Giancarlo Succi.
- Bossi, Piergiuliano, and Francesco Cirillo. 2001. Repo Margining System: Applying XP in the Financial Industry. *Proceedings of the 2nd International*

- Conference on Extreme Programming and Flexible Processes in Software Engineering* (XP2001).
- Brenner, Lyle A., Derek J. Koehler, and Amos Tversky. 1996. On the Evaluation of One-sided Evidence. *Journal of Behavioral Decision Making* 9: 59–70.
- Brooks, Fred. 1975. *The Mythical Man Month: Essays on Software Engineering*. Addison-Wesley.
- Center for Quality of Management. Special Issue on Kano's Methods for Understanding Customer-defined Quality. *Center for Quality of Management Journal* 2 (4).
- Cirillo, Francesco. 2005. Tracking the Project with the Pomodoro. Unpublished article.
- Clark, Kim B., and Steven C. Wheelwright. 1993. *Managing New Product and Process Development: Text and Cases*. The Free Press.
- Cockburn, Alistair. 2002. Agile Software Development Joins the "Would-Be" Crowd. *Cutter IT Journal* 15 (1).
- Cohn, Mike. 2004. *User Stories Applied: For Agile Software Development*. Addison-Wesley.
- Constantine, Larry L., and Lucy A. D. Lockwood. 1999. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley.
- DeGrace, Peter, and Leslie Stahl. 1990. *Wicked Problems, Righteous Solutions: A Catalog of Modern Engineering Paradigms*. Prentice Hall.
- DeLuca, Jeff. 2002. *FDD Implementations*.
www.nebulon.com/articles/fdd/fddimplementations.html.
- DeMarco, Tom, and Timothy Lister. 2003. *Waltzing with Bears: Managing Risk on Software Projects*. Artech House.
- Duggan, Jim, Jason Byrne, and Gerald J. Lyons. 2004. A Task Allocation Optimizer for Software Construction. *IEEE Software* 21 (3): 76–82.
- Ganssle, Jack. 2004. The Middle Way. *Embedded Systems Programming*, October 14.
- van Genuchten, Michiel. 1991. Why Is Software Late? An Empirical Study of Reasons for Delay in Software Development. *IEEE Transactions on Software Engineering* 17 (6): 582–590.
- Gilb, Tom. 1988. *Principles of Software Engineering Management*. Addison-Wesley.
- Githens, Greg. 1998. Rolling Wave Project Planning. *Proceedings of the 29th Annual Project Management Institute 1998 Seminars and Symposium*.
- Goldratt, Eliyahu M. 1990. *What Is This Thing Called Theory of Constraints and How Should It Be Implemented?* North River Press.
- Goldratt, Eliyahu M. 1992. *The Goal: A Process of Ongoing Improvement*, 2nd rev. ed. NorthRiver Press.
- Goldratt, Eliyahu M. 1997. *Critical Chain*. North River Press.

- Goldratt, Eliyahu M. 2000. *Necessary But Not Sufficient*. North River Press.
- Grenning, James. 2002. Planning Poker, www.objectmentor.com/resources/articles/PlanningPoker.zip.
- Griffin, Abbie, and John R. Hauser. 1993. The Voice of the Customer. *Marketing Science* 12 (1): 1–27.
- Hagafors, R., and B. Brehmer. 1983. Does Having to Justify One's Decisions Change the Nature of the Decision Process? *Organizational Behavior and Human Performance* 31: 223–232.
- Highsmith, Jim. 2004a. *Agile Project Management: Creating Innovative Products*. Addison Wesley.
- Highsmith, Jim. 2004b. Email to agilemanagement@yahoogroups.com on February 16, 2004.
- Hobbs, Charles. 1987. *Time Power*. Harper & Row Publishers.
- Hoest, Martin, and Claes Wohlin. 1998. An Experimental Study of Individual Subjective Effort Estimations and Combinations of the Estimates. *Proceedings of the 20th International Conference on Software Engineering*, pp. 332–339.
- Hunt, Andrew, and David Thomas. 1999. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley.
- Jeffries, Ron. 2004. Email to extremeprogramming@yahoogroups.com on July 1, 2004.
- Jeffries, Ron, Ann Anderson, and Chet Hendrickson. 2001. *Extreme Programming Installed*. Addison-Wesley.
- Johnson, Jim. 2002. Keynote speech at Third International Conference on Extreme Programming.
- Johnson, Philip M., Carleton A. Moore, Joseph A. Dane, and Robert S. Brewer. 2000. Empirically Guided Software Effort Estimation. *IEEE Software* 17 (6): 51–56.
- Jørgensen, Magne. 2004. A Review of Studies on Expert Estimation of Software Development Effort. *Journal of Systems and Software* 70 (1–2): 37–60.
- Jørgensen, Magne, and Kjetil Moløkken. 2002. Combination of Software Development Effort Prediction Intervals: Why, When and How? *Fourteenth IEEE Conference on Software Engineering and Knowledge Engineering*.
- Keen, Jack M., and Bonnie Digrius. 2003. *Making Technology Investments Profitable: ROI Roadmap to Better Business Cases*. John Wiley & Sons.
- Kennedy, Michael N. 2003. *Product Development for the Lean Enterprise: Why Toyota's System Is Four Times More Productive and How You Can Implement It*. The Oaklea Press.
- Kernighan, Brian, and P. J. Plauger. 1974. *The Elements of Programming Style*. McGraw-Hill.
- Larson, Carl E., and Frank M. J. LaFasto. 1989. *Teamwork: What Must Go Right / What Can Go Wrong*. SAGE Publications.
- Laufer, Alexander. 1996. *Simultaneous Management: Managing Projects in a*

- Dynamic Environment*. American Management Association.
- Leach, Lawrence P. 2000. *Critical Chain Project Management*. Artech House.
- Lederer, Albert L., and Jayesh Prasad. 1992. Nine Management Guidelines for Better Cost Estimating. *Communications of the ACM* 35 (2): 51–59.
- Lederer, Albert L., and Jayesh Prasad. 1998. A Causal Model for Software Cost Estimating Error. *IEEE Transactions on Software Engineering* 24(2):137–148.
- McClelland, Bill. 2004. Buffer Management. *The TOC Times*, February.
- McConnell, Steve. 1998. *Software Project Survival Guide*. Microsoft Press.
- Macomber, Hal. 2004. Achieving Change in Construction Is a Matter of Mental Models. *Reforming Project Management* 3 (35): August 29.
- Malotaux, Niels. 2004. Evolutionary Project Management Methods. www.malotaux.nl/nrm/pdf/MxEvo.pdf.
- Martin, Robert C. 2004. PERT: Precursor to Agility. *Software Development*, February.
- Miranda, Eduardo. 2001. Improving Subjective Estimates Using Paired Comparisons. *IEEE Software* 18 (1): 87–91.
- Mugridge, Rick, and Ward Cunningham. 2005. *FIT for Developing Software: Framework for Integrated Tests*. Prentice Hall.
- Nejmeh, Brian A., and Ian Thomas. 2002. Business-Driven Product Planning Using Feature Vectors and Increments. *IEEE Software* 34 (6): 34–42.
- Newbold, Robert C. 1998. *Project Management in the Fast Lane: Applying the Theory of Constraints*. St. Lucie Press.
- Parkinson, Cyril Northcote. 1958. *Parkinson's Law: The Pursuit of Progress*. John Murray.
- Poppendieck, Mary and Tom. 2003. *Lean Software Development: An Agile Toolkit*. Addison-Wesley.
- Poppendieck, Mary. 2003. Lean Development and the Predictability Paradox. *Agile Software Development and Project Management Executive Report* 4 (8). Cutter Consortium.
- Putnam, Lawrence H., and Ware Myers. 1997. How Solved Is the Cost Estimation Problem? *IEEE Software* 14 (6): 105–107.
- Rakitin, Steven R. Creating Accurate Estimates and Realistic Schedules. *Software Quality Professional* 4 (2): 30–36.
- Reinertsen, Donald. G. 1997. *Managing the Design Factory: A Product Developers's Toolkit*. Free Press.
- Rising, Linda. 2002. Agile Meetings. *Software Testing and Quality Engineering* 4 (3): 42–46.
- Saaty, Thomas. 1996. *Multicriteria Decision Making: The Analytic Hierarchy Process*. RWS Publications.
- Sanders, G. S. 1984. Self Presentation and Drive in Social Facilitation. *Journal of Experimental Social Psychology* 20 (4): 312–322.
- Sauerwein, Elmar, Franz Bailom, Kurt Matzler, and Hans H. Hinterhuber. 1996.

- The Kano Model: How to Delight Your Customers. *International Working Seminar on Production Economics* 313–327.
- Schwaber, Ken, and Mike Beedle. 2002. *Agile Software Development with Scrum*. Prentice Hall.
- Smith, Preston G., and Donald G. Reinertsen. 1997. *Developing Products in Half the Time: New Rules, New Tools*, 2nd ed. Wiley.
- Standish Group International, Inc., The. 2001. *Extreme Chaos*.
www.standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf.
- Takeuchi, Hirotaka, and Ikujiro Nonaka. 1986. The New New Product Development Game. *Harvard Business Review*, January.
- Tockey, Steve. 2004. *Return on Software: Maximizing the Return on Your Software Investment*. Addison-Wesley.
- Van Schooenderwoert, Nancy J. 2004. Interview with Ward Cunningham and Scott Densmore. *Agile Times* 6: 61–69.
- Vicinanza, S., T. Mukhopadhyay, and M. J. Prietula. 1991. Software Effort Estimation: An Exploratory Study of Expert Performance. *Information Systems Research* 2 (4): 243–262.
- Weinberg, Gerald M., and E. L. Schulman. 1974. Goals and Performance in Computer Programming. *Human Factors* 16 (1): 70–77.
- Wiegers, Karl. 1999. First Things First: Prioritizing Requirements. *Software Development*, September.

[1] Не забывайте, что \$ — это универсальный, обобщенный символ для обозначения валюты.

[2] Чтобы рассчитать расстояние до горизонта в километрах, умножьте корень квадратный из высоты ваших глаз над уровнем воды на 3,57.

[3] Такого определения в данный момент достаточно. В следующей главе мы будем говорить об идее оценки в идеальных днях как об альтернативе оценки в пунктах. Команда, которая осуществляет оценку в идеальных днях, определяет скорость как сумму оценок реализованных историй в идеальных днях.

[4] Каждое число в последовательности Фибоначчи представляет собой сумму двух предыдущих чисел.

[5] Автор книги по бережливой разработке программного обеспечения.
— *Прим. пер.*

[6] Технически мне нужно разделить ее на 0,6 и на 1,6. Однако из-за того, что 0,6 и 1,6 представляют собой взаимно обратные величины ($0,6 \times 1,6 = 0,96 \approx 1$), примерно те же самые величины получаются при умножении.

[7] Определение размера буфера подобным образом ранее предлагали Райнертсен (Reinertsen, 1997), Ньюболд (Newbold, 1998) и Лич (Leach, 2000).

[8] Именно поэтому этот метод определения размера буфера обычно называют «квадратный корень из суммы квадратов».

Переводчик *В. Ионов*

Главный редактор *С. Турко*

Руководитель проекта *А. Василенко*

Корректоры *Е. Аксёнова, О. Улантикова*

Дизайн обложки *Ю. Буга*

Компьютерная верстка *А. Абрамов*

© Authorized translation from the English language edition, published by Pearson Education, Inc.; publishing as Prentice Hall. Copyright © 2006 by Pearson Education, Inc. All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

© Издание на русском языке, перевод, оформление. ООО «Альпина Паблишер», 2018 год

© Электронное издание. ООО «Альпина Диджитал», 2018

Кон М.

Agile: оценка и планирование проектов / Майк Кон; Пер. с англ. — М.: Альпина Паблишер, 2018.

ISBN 978-5-9614-5208-2