
БИБЛИОТЕКА ПРОГРАММИСТА
Кент Бек
Экстремальное программирование
экономические аспекты программного обеспечения
стоимость внесения изменений
базовые принципы
стратегия менеджмента
организация рабочего места
разделение полномочий
планирование
адаптация для существующего проекта
жизненный цикл идеального проекта
Addison Wesley
ПИТЕР


Кент Бек
БИБЛИОТЕКА ПРОГРАММИСТА
Экстремальное программирование
Экстремальное программирование — это упрощенная методика организации производства для небольших и средних по размеру команд разработчиков, занимающихся созданием программного продукта в условиях нечетких или быстро меняющихся требований. Данная книга поможет определить, оправдано ли применение XP в вашей ситуации.
ПИТЕР
экономические аспекты программного обеспечения
стоимость внесения изменений
базовые принципы
стратегия менеджмента
организация рабочего места
разделение полномочий
планирование
разработка
проектирование
тестирование
адаптация для существующего проекта
жизненный цикл идеального проекта
Уровень пользователя: опытный/эксперт
Серия: Библиотека программиста

Краткое содержание

Содержание

Часть 3. Реализация XP

Посвящается моему отцу,

а также благодарю Синди Андрес (Cindee Andres), мою жену и партнера, за то, что она требовала, чтобы я не обращал на нее внимания и писал. Спасибо Бетани (Bethany), Линкольну (Lincoln), Форресту (Forrest) и Джолле (Joelle) за то, что они не отвлекали меня и предоставили мне возможность писать.

О серии XP

Экстремальное программирование (Extreme Programming), часто обозначаемое аббревиатурой XP, — это дисциплина разработки программного обеспечения и ведения бизнеса в области создания программных продуктов, которая фокусирует усилия обеих сторон (программистов и бизнесменов) на общих, вполне достижимых целях. Команды, использующие XP, производят качественное программное обеспечение с весьма большой скоростью. Методики, которые входят в состав дисциплины XP, описанной в данной книге, выбраны из-за того, что они основаны на человеческом творчестве и принятии того, что человек является существом неустойчивым и подверженным ошибкам.

XP часто представляется как набор методик, однако сама по себе XP не является финишной линией. Вам не надо все лучше и лучше практиковать и развивать XP для того, чтобы в конце этого процесса получить долгожданную золотую звезду. Напротив, XP — это линия старта. XP ставит вопрос: «Насколько минимальными могут быть наши усилия для того, чтобы мы могли продолжать производить качественное программное обеспечение?»

Начало ответа на вопрос звучит так: если мы хотим разрабатывать качественные программы без суматохи и путаницы, мы должны быть готовыми целиком и полностью внедрить у себя в команде несколько методик, которые мы собираемся использовать в полной мере. Если мы будем использовать эти методики наполовину, проблемы останутся и, чтобы их решить, необходимо будет перейти к использованию методик в полной мере. Если мы ограничимся полумерами, с течением времени мы в них запутаемся настолько, что не сможем понять, что то основное, что создается трудом программистов, возникает на свет благодаря программированию.

Я сказал «начало ответа на вопрос...», так как продолжения на самом деле не существует. Люди, создававшие и внедрявшие XP, тоже думали над решением этого вопроса. Попробовав использовать XP, они перешагнули порог и побывали в неизведанном. Вернувшись, они рассказали свою историю. Изложенные ими мысли — это указатели, расставленные вдоль дороги: «Здесь живут драконы», «Через 15 км открывается хороший вид», «Этот участок опасен во время дождя».

Прошу прощения, но мне пора идти программировать.

Кент Бек, консультант серии

Предисловие

Экстремальное программирование (eXtreme Programming, XP) определяет кодирование как ключевую и основополагающую деятельность при работе над программным проектом. Возможно, что это неправильно!

Думаю, что стоит вспомнить о моем собственном опыте разработки программного обеспечения. Я работаю в среде, где разрабатываемый продукт постоянно находится в работоспособном состоянии, и при этом в него постоянно вносятся изменения. Сроки выпуска очередной работоспособной версии чудовищно сжаты, и при этом над всем этим нависает огромный технический риск. В подобной среде способность поправить своего соратника — это искусство, без которого не выжить. Обмен информацией как внутри некоторой команды, так и между несколькими командами, которые часто разделены географически, выполняется при помощи кода. Мы читаем код для того, чтобы понять устройство новых или модифицированных программных интерфейсов системы. Жизненный цикл и поведение сложных объектов определяются с использованием тестовых случаев, то есть снова при помощи кода. Сообщения о возникающих проблемах сопровождаются тестовыми случаями, демонстрирующими проблему, для этого опять используется код. Наконец, мы постоянно заняты улучшением существующего кода, делая его более производительным, более гибким, более понятным. Очевидно, что в подобных условиях разработка программного продукта почти целиком основана на кодировании, однако при этом нам удается с успехом завершать проекты к сроку, таким образом, данный подход вполне жизнеспособен.

Не следует делать вывод, что все, что вам потребуется для успешной реализации программного проекта, — это безоглядное ожесточенное программирование. Разрабатывать программное обеспечение очень непросто, а разрабатывать качественное программное обеспечение и при этом завершать работу в срок — еще сложнее. Чтобы описанный мною подход сработал, необходимо последовательное применение важных дополнительных правил и методик. Именно с этого Кент Бек (Kent Beck) начинает свою побуждающую к размышлениям книгу об XP.

Кент был среди тех руководителей компании Tektronix, которые осознали огромный потенциал, заложенный в методике программирования в связанных парах при разработке сложных инженерных приложений

в среде Smalltalk. Вместе с Вардом Каннингхемом (Ward Cunningham) Кент стал вдохновителем развития методики программирования по образцам (ее еще называют программированием с использованием *паттернов — patterns programming*[1]), которая в значительной степени повлияла на мою собственную карьеру. В рамках XP описывается подход к разработке программного обеспечения, который сочетает в себе методики, используемые многочисленными успешно работающими разработчиками, которые изучили множество литературы, посвященной организации труда программистов, и опробовали на практике множество методов и процедур разработки программного продукта. Как и программирование по образцам, XP формирует набор наиболее эффективных методик, таких как тестирование программных модулей, программирование парами и переработка кода. В рамках XP эти методики скомбинированы таким образом, чтобы дополнять и часто контролировать друг друга. Основная цель данной книги — рассказать о взаимодействии и совместном использовании различных методик. У всех методик программирования одна цель — создание программного продукта с заданной функциональностью к заданному сроку. Предлагаемый OTI весьма успешный процесс *Just In Time Software* не является XP в чистом виде, однако между этими двумя подходами очень много общего.

Я сотрудничал с Кентом и использовал описанные в рамках XP эпизоды при работе над небольшим, но небезызвестным проектом под названием JUnit[2]. Его взгляды и подходы к разработке программ всегда заставляли меня задумываться над тем, как лично я привык работать над программным проектом. Без сомнения, XP ставит под вопрос многие традиционные подходы, используемые в индустрии программирования. Прочитав данную книгу, вы сможете самостоятельно решить, надо ли вам применять XP в своей работе или нет.

Эрих Гамма (Erich Gamma)

[1] См. книгу Э. Гамма, Р. Хельма, Р. Джонсона и Дж. Влиссидеса «Приемы объектно-ориентированного проектирования. Паттерны проектирования», выпущенную издательствами «Питер» и «ДМК» в 2001 году. — *Примеч. ред.*

[2] JUnit — программная инфраструктура, предназначенная для автоматического тестирования модулей в среде Java. — *Примеч. пер.*

Введение

Эта книга об экстремальном программировании (eXtreme Programming, XP). Экстремальное программирование — это упрощенная методика организации производства для небольших и средних по размеру команд специалистов, занимающихся разработкой программного продукта в условиях неясных или быстро меняющихся требований. Данная книга предназначена для того, чтобы помочь определить, оправдано ли применение XP в вашей ситуации.

Для многих XP выглядит набором вполне приемлемых и оправданных с точки зрения здравого смысла методов организации труда. Тогда почему программирование в соответствии с методиками XP называется экстремальным? Дело в том, что XP доводит использование многих общепринятых и широко используемых принципов программирования до экстремальных уровней.

- Если пересмотр кода — это хорошо, значит, мы будем пересматривать код постоянно (программирование парами);
- если тестирование — это хорошо,значит, каждый участник проекта будет тестировать код программы постоянно (тестирование модулей), даже заказчики (функциональное тестирование);
- если проектирование — это хорошо, значит, проектирование надо сделать составной частью повседневной работы каждого из участников проекта (переработка кода);
- если простота — это хорошо, значит, мы должны сохранять в системе наиболее простой дизайн, обеспечивающий текущий требуемый уровень функциональности (наиболее простая вещь которая скорее всего сработает);
- если архитектура важна, значит, каждый из участников проекта будет постоянно работать над определением и пересмотром архитектуры (метафора);
- если интеграционное тестирование важно, значит, необходимо собирать и тестировать разрабатываемую систему несколько раз на дню (продолжающаяся интеграция);

- если небольшие итерации — это хорошо, необходимо сделать итерации очень, очень маленькими — секунды, минуты, может быть часы, но не недели и месяцы, и ни в коем случае не годы (игра в планирование).

Когда я впервые решил сформулировать для себя суть XP, я представил себе набор рукояток на пульте управления. Каждая рукоятка соответствовала некоторой методике, о которой из своего личного опыта я знал, что она вполне эффективна. Каждая рукоятка позволяла использовать ту или иную методику в определенной степени: от 1 до 10. Я попробовал установить все рукоятки в максимально возможное положение (10) и с удивлением обнаружил, что полный набор рассматриваемых мной методик остается стабильным, предсказуемым и гибким.

XP формирует два набора обещаний:

- Программистам XP обещает, что каждый из них будет работать над решением действительно важных задач каждый рабочий день. Каждый из них никогда не окажется в затруднительном положении в одиночку. Каждый из них будет способен сделать все от него зависящее для того, чтобы сделать разрабатываемую систему удачной. Каждый из них будет способен принять решение именно в той области, в которой он компетентен, и если в некоторой области он не достаточно компетентен, он не будет участвовать в принятии решения.
- Заказчикам и менеджерам XP обещает, что они получат максимальную возможную отдачу от каждой недели работы над проектом. Каждые несколько недель они будут способны увидеть прогресс в достижении заданных ими целей. Они получат возможность изменить направление развития проекта в самой середине разработки, не опасаясь при этом дополнительных экстраординарных затрат.

Если говорить коротко, XP обещает снизить связанный с проектом риск, улучшить реакцию на изменение бизнеса, улучшить производительность работы над проектом и сделать процесс разработки программного обеспечения более приятным — и все это в одно и то же время. Я не шучу, хватит смеяться. Просто прочитайте эту книгу, и вы сами сможете проверить, сошел ли я с ума.

Данная книга

Данная книга рассказывает о вещах, связанных с методикой XP, — ее корнях, философии, разного рода историях и мифах. Книга предназначена для того, чтобы помочь вам принять взвешенное решение о том, надо ли

использовать XP в вашем собственном проекте. Если, прочитав данную книгу, вы решили *не* использовать XP при работе над своим проектом, я буду считать основную цель достигнутой точно так же, как если бы, прочитав данную книгу, вы приняли бы решение о том, что XP — это то, что вам нужно. Вторая цель книги — помочь тем из вас, кто уже использует XP. Прочитав книгу, такие читатели смогут лучше понять эту методику.

Эта книга не содержит точных инструкций о том, как именно должен осуществляться процесс экстремального программирования. Вы не увидите в ней множества примеров, алгоритмов или программистских историй. Для того чтобы получить все это, вы можете поискать в Интернете, поговорить с некоторыми из участников проектов, подождать появления посвященных этому книг или сами заняться созданием подобного материала.

Дальнейшая судьба описанной мною дисциплины разработки программного обеспечения XP находится в руках группы людей (возможно вы являетесь одним из них), которые неудовлетворены существующими на данный момент традиционными методиками организации труда программистов. Вы нуждаетесь в лучшем способе разработки программного обеспечения, вы хотите наладить более продуктивные и доброжелательные отношения с вашими заказчиками, вы хотите сделать работающих под вашим руководством программистов более счастливыми, более лояльными и более производительными. Короче говоря, вы желаете получить значительное преимущество и вы не боитесь воспользоваться новыми идеями для того, чтобы обрести это преимущество. Однако, прежде чем пойти на риск, вы хотите убедиться в том, что вы по крайней мере не полный дурак.

XP предписывает вам делать работу совсем иначе, чем вы к этому привыкли. В некоторых случаях рекомендации XP совершенно противоречат общепризнанным нормам. На текущий момент я полагаю, что воспользоваться XP смогут только те, у кого есть весомые причины изменить существующий порядок вещей. Однако если такие причины существуют, вы можете приступать к использованию XP прямо сейчас. Я написал эту книгу для того, чтобы вы смогли узнать об этих причинах подробнее.

Что такое XP?

Что такое XP? XP — это упрощенный, эффективный, гибкий, предсказуемый, научно обоснованный и весьма приятный способ разработки программного обеспечения, предусматривающий низкий уровень риска. От других методик XP отличается по следующим признакам.

- Благодаря использованию чрезвычайно коротких циклов разработки XP предлагает быструю, реальную и постоянно функционирующую обратную связь.

- В рамках XP используется планирование по нарастающей, в результате общий план проекта возникает достаточно быстро, однако при этом подразумевается, что этот план эволюционирует в течение всего времени жизни проекта.
- В рамках XP используется гибкий график реализации той или иной функциональности, благодаря чему улучшается реакция на изменение характера бизнеса и меняющиеся в связи с этим требования заказчика.
- XP базируется на автоматических тестах, разработанных как программистами, так и заказчиками. Благодаря этим тестам удается следить за процессом разработки, обеспечивать корректное эволюционирование системы и без промедления обнаруживать существующие в системе дефекты.
- XP основана на оральном обмене информацией, тестах и исходном коде. Три этих инструмента используются для обмена сведениями о структуре системы и ее поведении.
- XP базируется на процессе эволюционирующего дизайна, который продолжается столь же долго, сколько существует сама система.
- XP базируется на тесном взаимодействии программистов, обладающих самыми обычными навыками и возможностями.
- XP основывается на методиках, которые удовлетворяют как краткосрочным инстинктам отдельных программистов, так и долгосрочным интересам всего проекта в целом.

XP — это дисциплина разработки программного обеспечения. Это дисциплина потому, что в рамках XP существуют определенные вещи, которые вы обязаны делать, если вы намерены использовать XP. Вы не должны выбирать, надо или не надо писать тесты, потому что если вы этого не делаете, программирование, которым вы занимаетесь, нельзя назвать экстремальным: конец дискуссии.

Методика XP предназначена для работы над проектами, над которыми может работать от двух до десяти программистов, которые не зажаты в жесткие рамки существующего компьютерного окружения и в которых вся необходимая работа, связанная с тестированием, может быть выполнена в течение одного дня.

XP пугает или раздражает некоторых людей, которые сталкиваются с этой методикой впервые. Вместе с тем ни одна идея, лежащая в основе XP, не является новой. В некотором смысле методика XP консервативна — все используемые в ее рамках приемы проверены десятилетиями (что касается стратегии реализации) и даже столетиями (что касается стратегии менеджмента) практики.

Нововведениями в XP являются следующие особенности:

- все эти давно известные приемы собраны под одной крышей;
- интенсивность, с которой эти приемы внедряются в повседневную работу, доведена до крайности;
- используемые методики поддерживают одна другую в наибольшей возможной степени.

Достаточность

В своих работах *The Forest People* «Лесные народы» и *The Mountain People* «Горные народы» антрополог Колин Тернбулл (Colin Turnbull) описывает два совершенно отличающихся друг от друга общества. В горах необходимых для жизни ресурсов не хватает, и люди всегда находятся на грани голода. Культура, которая возникла в подобных условиях, выглядит ужасающе. Матери бросают своих детей ради того, чтобы выжить самим. Пропитание может стать причиной смертоубийства. Жестокость, зверство и предательство являются обыденными и повседневными.

В отличие от гор лес насыщен ресурсами. Чтобы обеспечить себя пропитанием на целый день, человеку достаточно потратить всего около получаса. Лесная культура является обратным отражением горной культуры. Взрослые принимают участие в выращивании и воспитании общих детей, которые растут в любви и заботе до тех пор, пока не становятся достаточно дееспособными, чтобы позаботиться о себе самостоятельно. Если один человек по оплошности убивает другого (преднамеренное убийство этим людям незнакомо), его изгоняют из общества. Однако при этом изгнанник просто должен удалиться в лес и провести там несколько месяцев, и даже тогда некоторые члены племени приносят ему еду и подарки.

XP — это попытка ответить на вопрос: «Как лично вы программировали бы, если бы у вас было достаточно времени?» Вообще-то у вас нет лишнего времени, так как в конце концов программирование — это бизнес. В любом бизнесе побеждает тот, кто делает работу быстрее. Однако если бы у вас было время, вы наверняка уделяли бы внимание разработке тестов; вы наверняка заново переделали бы архитектуру системы в случае, если пришли бы к выводу, что это необходимо; вы наверняка больше бы общались со своими соратниками-программистами, а также с заказчиком.

Подобное «ощущение достаточности» выглядит более человечным в отличие от ситуаций, когда программисты из последних сил пытаются удержаться в заданных временных рамках, выбиваются из графика, не успевают выполнить большой объем чрезвычайно важной работы лишь для того, чтобы успеть сдать проект в срок. Спешка мешает программис-

там в полной мере проявить свой талант и получить удовольствие от работы. Однако приступая к изучению XP, вы должны понять, что ощущение достаточности — это тоже неплохой бизнес. Ощущение достаточности становится источником эффективности точно так же, как ощущение недостатка рождает сбои в работе, ведет к появлению дефектов и снижению качества и, в конечном итоге, снижению производительности труда.

План книги

Книга написана так, как будто вы и я вместе занимаемся созданием новой дисциплины разработки программного обеспечения. Мы начинаем с изучения наших базовых представлений о разработке программного обеспечения. После этого мы собственно создаем новую дисциплину. Затем мы изучаем последствия того, что мы создали, — как это может быть внедрено и использовано на практике, когда это не следует использовать и какие возможности это открывает перед бизнесом.

Книга разделена на три части.

- *Проблема*: в главах, начиная с *«Риск: основная проблема»* и заканчивая *«Обратно к истокам»*, определяется проблема, которую пытается решить экстремальное программирование, а также устанавливаются критерии, благодаря которым можно оценить качество решения. В данной части вы получаете общее представление о методике экстремального программирования.
- *Решение*: в главах, начиная с *«Краткий обзор»* и заканчивая *«Стратегия тестирования»*, абстрактные идеи, представленные в первой части книги, превращаются в набор методик конкретной дисциплины. В данной главе не содержится каких-либо сведений о том, как именно вы можете применить описанные методики на практике. Вместо этого речь идет об общей форме каждой из методик. Рассуждая о каждой методике, я связываю ее с проблемами и принципами, обсуждавшимися в первой части книги.
- *Реализация XP*. в главах, начиная с *«Внедрение XP»* и заканчивая «XP *в работе»,* обсуждается множество вопросов, связанных с внедрением XP, — как следует внедрить XP, чего следует ожидать от разных людей, участвующих в проекте XP, какое представление об XP складывается у людей делового мира.

Благодарности

Я обращаюсь к читателям от первого лица не потому, что в книге представлены мои собственные идеи, а потому, что я рассказываю вам о моем

собственном восприятии этих идей. Большая часть методик, используемых в рамках XP, являются старыми, как все программирование.

Вард Каннингхэм (Ward Cunningham) — это мой основной источник, из которого я черпал излагаемый в книге материал. Так или иначе я потратил последние пятнадцать лет, просто пытаясь объяснить другим людям то, чем Вард уже давно занимается практически. Спасибо также Рону Джеффрису (Ron Jeffries) за то, что он тоже это попробовал, а затем существенно улучшил. Спасибо Мартину Фолеру (Martin Fowler) за то, что он объясняет все это простым мягким языком и без нервных срывов. Спасибо Эриху Гамма (Erich Gamma) за длительные беседы, совмещенные с созерцанием лебедей в Лимме, а также за то, что он не позволил мне покинуть его с плохими мыслями в голове. И конечно же, ничего этого не произошло бы в моей жизни, если бы у меня не было такого примера для подражания, как мой отец, Дуг Бек (Doug Beck), который оттачивал свое мастерство программирования в течение многих лет.

Спасибо команде C3 в компании Chrysler за то, что они сопровождали меня в моих изысканиях. А также особое спасибо нашим менеджерам Сью Ангер (Sue Unger) и Рону Сэведжу (Ron Savage) за то, что у них хватило храбрости дать нам попробовать.

Спасибо компании Daedalos Consulting за помощь в написании данной книги.

Чемпионская награда за просмотр материала переходит в руки Пола Чишолма (Paul Chisholm) за его обильные, емкие, скрупулезные, а зачастую откровенно раздражающие комментарии. Без его помощи данная книга не стала бы и наполовину столь же популярной.

Я действительно получал огромное удовольствие от общения со всеми теми, кто осуществлял предварительный просмотр того, что я написал. Их работа стала для меня огромным подспорьем. Не могу подыскать слова благодарности за то, что у них хватило терпения прочитать до конца всю эту прозу, для многих из них изложенную на иностранном языке. Спасибо (перечисление в случайном порядке, в котором я получал от них комментарии) Грегу Хатчинсону (Greg Hutchinson), Массимо Арнольди (Massimo Arnoldi), Дэйву Клилу (Dave Cleal), Сэмесу Шустеру (Sames Schuster), Дону Уелсу (Don Wells), Джошу Киревски (Joshua Kerievsky), Торстену Диттмару (Thorsten Dittmar), Морицу Бекеру (Moritz Becker), Дэниэлу Габлеру (Daniel Gubler), Кристофу Хенрики (Christoph Henrici), Томасу Зангу (Thomas Zang), Дирку Коэнигу (Dierk Koenig), Мирославу Новаку (Miroslav Novak), Роднёю Райану (Rodney Rayan), Френку Вестфалу (Frank Westphal), Полу Трунцу (Paul Trunz), Стиву Хайесу (Steve Hayes), Кевину Бредтке (Kevin Bradtke), Джанин Де Гузман (Jeanine De Guzman), Тому Кабиту (Tom Kubit), Фалку Бругманну (Falk Bruegmann),

Хаско Хейнеке (Hasko Heinecke), Петеру Мерелу (Peter Merel), Робу Ми (Rob Mee), Пете Макбрину (Pete McBreen), Томасу Эрнсту (Thomas Ernst), Гуидо Хечлеру (Guido Haechler), Дитеру Холцу (Dieter Holz), Мартину Кнехту (Martin Knecht), Дирку Крампе (Dierk Krampe), Патрику Лиссеру (Patrick Lisser), Элизабет Майер (Elisabeth Maier), Томасу Мансини (Thomas Mancini), Алексио Морено (Alexio Moreno), Рольфу Пфеннингеру (Rolf Pfenninger) и Мэтиасу Рёсселу (Matthias Ressel).

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Все исходные тексты, приведенные в книге, вы можете найти по адресу http://www.piter.com/download.

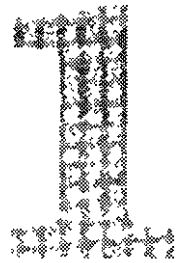
На web-сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

Часть 1

Проблема

В данной части книги подготавливается сцена, на которой должно в последующем появиться экстремальное программирование. Здесь описываются различные аспекты проблемы, которую предстоит решить, сформировав новую дисциплину разработки программного обеспечения. В данной части обсуждаются базовые предположения, которые мы должны учитывать, подбирая методики разработки программного обеспечения, — метафора управления автомобилем, четыре значения, принципы, сформированные на основе этих значений, а также деятельность, которую требуется структурировать в рамках новой дисциплины разработки программного обеспечения.

Риск: основная проблема



Существующие дисциплины разработки программного обеспечения не срабатывают и не дают желаемого экономического эффекта. Эта проблема обладает огромным экономическим и гуманитарным значением. Мы нуждаемся в новом способе разработки программного обеспечения.

Основная проблема разработки программного обеспечения — это риск. Вот несколько примеров риска.

- Смещение графиков — наступает день сдачи работы, и вы вынуждены сообщить заказчику, что разрабатываемая система не будет готова еще в течение шести месяцев.
- Закрытие проекта — после нескольких смещений графика и переносов даты сдачи проект закрывается, даже не будучи доведен до стадии опробования в рабочих условиях.
- Система теряет полезность — разработанное программное обеспечение успешно устанавливается в реальной производственной рабочей среде, однако после пары лет использования стоимость внесения в нее изменений и/или количество дефектов увеличиваются настолько, что становится дешевле заменить систему новой разработкой.
- Количество дефектов и недочетов — программная система устанавливается в реальной производственной рабочей среде, однако количество дефектов и недочетов столь велико, что система не используется.
- Несоответствие решаемой проблеме — программная система устанавливается в реальной производственной рабочей среде, однако выясняется, что на самом деле она не решает проблему бизнеса, для решения которой она изначально предназначалась.

- Изменение характера бизнеса — программная система устанавливается в реальной производственной рабочей среде, однако в течение шести последних месяцев проблема, для решения которой предназначалась эта система, потеряла актуальность, а вместо нее бизнес столкнулся с новой, еще более серьезной проблемой.
- Недостаток возможностей — программная система обладает множеством потенциально интересных возможностей, каждую из которых было очень приятно программировать, однако выясняется, что ни одна из этих возможностей не приносит заказчику достаточно много пользы.
- Текучка кадров — в течение двух лет работы все хорошие программисты, работавшие над проектом, один за другим возненавидели разрабатываемую программную систему и ушли на другую работу.

На страницах данной книги вы прочитаете об экстремальном программировании (eXtreme Programming, XP) — дисциплине разработки программного обеспечения, которая ориентирована на снижение степени риска на всех уровнях процесса разработки. XP способствует существенному увеличению производительности и улучшению качества разрабатываемых программ, кроме того, это весьма занятная практика, доставляющая всем ее участникам массу удовольствия.

Каким образом XP снижает перечисленные ранее риски?

- Смещение графика — XP предлагает использовать очень короткие сроки выпуска каждой очередной версии. Предполагается, что каждая очередная готовая к использованию версия системы разрабатывается в течение максимум нескольких месяцев. Таким образом, объем работ в рамках каждой версии ограничен, а значит, если и происходит смещение, оно менее значительное. В рамках каждой версии предусматривается выпуск нескольких итераций запрашиваемых заказчиком возможностей, на разработку каждой из этих итераций уходит от одной до четырех недель. Так обеспечивается гибкая и чуткая обратная связь с заказчиком, благодаря чему он получает представление о текущем ходе работ. В рамках каждой итерации планирование в соответствии с XP осуществляется в терминах нескольких задач, которые необходимо решить для получения очередной итерации. На решение каждой из задач отводится от одного до трех дней. В результате команда может обнаруживать и устранять проблемы даже в процессе итерации. Наконец, XP подразумевает, что возможности с наивысшим приоритетом будут реализованы в первую очередь. Таким образом, любые возможности, которые не удалось реализовать в рам-

ках данной очередной версии программного продукта, обладают меньшим приоритетом.

- Закрытие проекта — в рамках XP заказчик должен определить наименьший допустимый набор возможностей, которыми должна обладать минимальная работоспособная версия программы, имеющая смысл с точки зрения решения бизнес-задач. Таким образом, программистам потребуется приложить минимальное количество усилий для того, чтобы заказчик понял, нужен ли ему этот проект или нет.
- Система теряет полезность — в рамках XP создается и поддерживается огромное количество тестов, которые запускаются и перезапускаются после внесения в систему любого изменения (несколько раз на дню), благодаря этому удается тщательно следить за качеством разрабатываемой программы. XP постоянно поддерживает систему в превосходном состоянии. Дефектам просто не дают накапливаться.
- Количество дефектов и недочетов — в рамках XP разрабатываемая система тестируется как программистами, создающими тесты для каждой отдельной разрабатываемой функции, так и заказчиками, которые создают тесты для каждой отдельной реализованной возможности системы.
- Несоответствие решаемой проблеме — в рамках XP заказчик является составной частью команды, которая работает над проектом. Спецификация проекта постоянно перерабатывается в течение всего времени работы над проектом, благодаря этому любые уточнения и открытия, о которых заказчик сообщает команде разработчиков, немедленно находят свое отражение в разрабатываемой программе.
- Изменение характера бизнеса — в рамках XP цикл работы над очередной версией программы существенно укорачивается. Таким образом, ко времени выхода очередной работоспособной версии программного продукта бизнес не успевает претерпеть существенных изменений. В процессе работы над очередной версией заказчик может попросить отказаться от разработки некоторой функциональности и вместо нее добавить в программный продукт другие, совершенно новые возможности. Команда разработчиков даже не обратит внимания на то, работает ли она над реализацией новых программных механизмов, или осуществляется разработка возможностей, определенных еще несколько лет назад.
- Недостаток возможностей — в рамках XP осуществляется реализация только наиболее высокоприоритетных задач.

- Текучка кадров — XP предлагает программистам брать на себя ответственность самостоятельно определять объем работы и время, необходимое для выполнения этой работы. Они получают возможность сравнить свои предварительные оценки с тем, что получилось на самом деле. В рамках XP содержатся правила, определяющие, кто именно имеет право делать предварительные оценки и изменять их. За счет этого существенно снижается вероятность того, что программист окажется в растерянности перед возложенной на него задачей, которую заведомо невозможно решить. XP стимулирует интенсивное общение между членами команды разработчиков, снижая ощущение одиночества, которое может возникнуть в случае, если программист не доволен работой, которую он делает. Наконец, в рамках XP явно определяется модель смены кадров. Новые члены команды постепенно берут на себя все большую и большую ответственность. В процессе этого они пользуются поддержкой друг друга, а также программистов, которые входят в состав команды уже давно.

Наша цель

Если мы признаем, что риск является основной проблемой, которую требуется решить, то в каком месте следует искать решение этой проблемы? Перед нами стоит задача сформировать стиль разработки программного обеспечения, который позволил бы существенно снизить все перечисленные риски. Кроме того, мы должны как можно понятнее объяснить эту дисциплину программистам, менеджерам и заказчикам. Также мы должны сформировать набор рекомендаций, позволяющих адаптировать данную дисциплину для конкретных локальных условий (другими словами — определить, что в этой дисциплине является незыблемым, а что можно варьировать в зависимости от конкретных обстоятельств).

Именно об этом пойдет речь в первой и второй частях книги. Шаг за шагом мы тщательно изучим каждый из аспектов проблемы, сформируем круг вопросов, на которые нам предстоит найти ответ, а затем мы решим проблему. Мы начнем с рассмотрения базовых предположений, а затем получим решение, которое будет диктовать, каким именно образом должна осуществляться разнообразная деятельность (планирование, тестирование, разработка, дизайн и внедрение), связанная с разработкой программного продукта.

Эпизод из программистской практики

День за днем программист выполняет одну и ту же последовательность **действий,** которую можно назвать программным проектом в миниатюре: он изучает **задачу,** четко связанную с возможностью продукта, в которой нуждается заказчик, затем он формирует набор необходимых тестов, реализует поставленную задачу, вылизывает код, а потом интегрирует его в систему. В каждом таком эпизоде содержатся все этапы, которые обычно составляют процесс разработки крупного программного продукта.

Для начала небольшой экскурс вперед, то есть туда, куда мы с вами движемся. Данная глава описывает живой процесс экстремального программирования так, как это происходит на практике. Речь пойдет об эпизоде разработки. Эпизод разработки — это фрагмент деятельности программиста, когда он реализует некоторую инженерную задачу (наименьший квант планирования) и интегрирует ее с остальной системой.

Я смотрю на мой набор карточек с задачами. На самой верхней из них написано: «Экспорт значения квартальных издержек на момент обращения к системе». Я вспоминаю, что на сегодняшнем утреннем совещании ты сообщил присутствующим, что завершил блок вычисления значения квартальных издержек. Я обращаюсь к тебе (моему гипотетическому соратнику) с вопросом, не найдется ли у тебя времени помочь мне с разработкой блока экспорта. Ты отвечаешь: «Конечно!» Это одно из основных правил XP: если к тебе обращаются с просьбой о помощи, ты обязан ответить «да». Таким образом мы становимся партнерами по программированию в паре.

В течение пары минут мы обсуждаем работу, которую ты выполнил вчера. Ты рассказываешь мне о добавленном тобою в систему коде, о бинарных значениях и о том, как выглядят соответствующие тесты. Мы отодвигаем мой монитор чуть назад, так как при этом эффективность парного программирования несколько увеличивается.

Ты спрашиваешь у меня: «Как выглядят тестовые случаи для этой задачи?»

Я отвечаю: «После выполнения экспортирующего кода значения в экспортированной записи должны соответствовать бинарным значениям».

Ты спрашиваешь: «Какие поля требуется заполнить?»

«Я не знаю, надо спросить у Эдди».

Мы отрываем Эдди от работы на тридцать секунд, и он рассказывает нам о пяти полях, которые связаны с квартальными издержками.

Мы обращаемся к изучению структуры некоторых существующих тестовых случаев, связанных с экспортом. Обнаруживается, что один из тестовых случаев почти полностью нам подходит. Если, используя обнаруженный нами код, создать абстрактный суперкласс, на его основе мы могли бы с легкостью реализовать наш тестовый случай. Так мы выполняем переработку кода (refactoring). Потом мы запускаем существующие тесты, и все они успешно выполняются.

После этого мы обращаем внимание, что несколько других тестовых случаев можно переработать так, чтобы они были основаны на только что созданном нами абстрактном суперклассе. Однако нам необходимо решить стоящую перед нами конкретную задачу, поэтому мы просто берем пустую карточку, записываем на ней «выполнить переработку тестовых случаев с использованием класса AbstractExportTest» и размещаем эту карточку в наборе задач, которые нам предстоит выполнить.

После этого мы разрабатываем наш тестовый случай. Благодаря тому, что перед этим мы разработали суперкласс, создание нового тестового случал не представляет труда. Мы выполняем эту работу в течение нескольких минут. Где-то в середине работы я говорю: «Я уже представляю себе, как мы можем это реализовать. Мы можем...»

Однако ты прерываешь меня: «Давай сначала завершим работу над тестовым случаем». Пока мы пишем тестовый случай, в голову приходят идеи для трех вариаций. Ты записываешь их на очередной карточке.

Мы завершаем работу над тестовым случаем и запускаем его. Он не срабатывает. Естественно, мы ведь еще ничего не реализовали. «Подожди минутку, — говоришь ты, — вчера, когда мы с Ральфом работали над блоком вычислений, мы написали пять тестовых случаев, о которых мы думали, что они не сработали. Все, за исключением одного, сработали с первого раза».

Мы запускаем отладчик и приступаем к изучению работы тестового случая. Мы анализируем объекты, с которыми имеет дело наш тестовый случай.

Я пишу код (или это делаешь ты — все зависит от того, у кого из нас возникнет более удачная идея). Пока мы работаем над реализацией, мы

замечаем еще пару дополнительных тестовых случаев, которые неплохо было бы написать. Мы заносим эту информацию на карточку. После того как код реализован, ранее разработанный тестовый случай срабатывает.

Мы переходим ко второму тестовому случаю, затем к третьему. Я реализую их. Ты обращаешь внимание, что код можно упростить. Мне с трудом удается одновременно слушать твои объяснения и продолжать писать код, поэтому я просто передаю тебе клавиатуру. Ты перерабатываешь код, после этого ты запускаешь тестовые случаи и они срабатывают. Ты приступаешь к реализации следующей пары тестовых случаев.

Спустя некоторое время мы смотрим на набор карточек с заданиями, которые необходимо доделать, и обнаруживаем, что нам осталось переделать тестовые случаи, которые были разработаны ранее. Все идет хорошо, и мы переделываем их, а затем убеждаемся в том, что все они срабатывают.

Теперь список дел, которые нам необходимо выполнить, пуст. Мы обращаем внимание на то, что компьютер, на котором выполняется интеграция, свободен. Мы загружаем последнюю версию системы, затем мы загружаем наши последние изменения. Затем мы запускаем все тестовые случаи, потом запускаем новые тестовые случаи, только что разработанные нами, а затем абсолютно все тестовые случаи, которые кто-либо когда-либо разрабатывал. Один из них не срабатывает. «Это очень странно, — произносишь ты, — прошло не меньше месяца с тех пор, когда я в последний раз сталкивался со сбоем тестового случая в процессе интеграции». Однако особых проблем в этом нет. Мы отлаживаем тестовый случай и исправляем код. После этого мы вновь запускаем весь полный набор тестов. На этот раз все они срабатывают. Мы включаем наш код в состав системы.

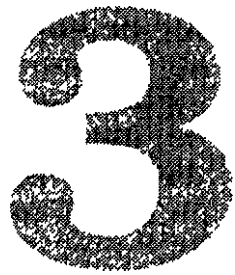
Вот так выглядит полный цикл разработки кода в рамках XP. Следует обратить внимание на следующие обстоятельства.

- Два программиста работают над решением задачи вместе в одной паре.
- Разработка базируется на тестах. Сначала вы пишете тесты, затем — код. Вы не можете считать работу доделанной до тех пор, пока все тесты не сработают. Если все тесты сработали и вы не в состоянии придумать еще каких-либо других тестов, которые могут не сработать, вы можете считать, что добавление функциональности завершено.
- Пары программистов не только добиваются того, чтобы тестовые случаи срабатывали. Они также выполняют эволюцию дизайна системы. Изменения не ограничиваются какой-то отдельной конкретно

очерченной областью. Пары программистов вносят свой вклад в анализ, дизайн, реализацию и тестирование всей системы. Добавление изменений осуществляется тогда, когда система нуждается в этих изменениях.

Сразу же за разработкой следует интеграция, которая включает в себя также интеграционное тестирование.

Экономика разработки программного обеспечения



Мы должны сделать разработку программного обеспечения экономически более выгодной, для этого мы должны тратить деньги **медленнее,** приносить прибыль **быстрее,** а также увеличивать продолжительность эффективного использования разрабатываемого нами программного продукта в реальных промышленных условиях. Но прежде всего мы должны обеспечить более широкую свободу для принятия **бизнес-решений**[1].

Сложив финансовые потоки, которые втекают в проект и вытекают из проекта, мы можем определить, за счет чего создается экономическая выгода, получаемая от проекта. Приняв во внимание коэффициенты прибыли (процентные ставки), мы можем вычислить чистый текущий объем финансовых потоков. После этого мы можем уточнить результаты нашего анализа, умножив скорректированный с учетом процентных ставок объем финансовых потоков на вероятность того, что проект выживет.

Для создания стратегии максимизации экономической выгоды, связанной с проектом, нам потребуется проанализировать следующие три фактора:

- финансовые потоки, втекающие в проект и вытекающие из проекта;
- коэффициенты прибыли (процентная ставка);
- вероятность того, что проект выживет.

Максимизировать экономическую выгоду можно следующим образом:

- мы можем тратить меньше, однако это достаточно сложно, так как изначально каждый обладает примерно одним и тем же набором знаний и инструментов;

Спасибо Джону Фаваро (John Favaro) за анализ XP с точки зрения ценовых вариаций.

- мы можем зарабатывать больше, однако это возможно только при наличии сверхсовершенной организации маркетинга и продаж, а эти темы в данной книге не обсуждаются (слава богу);
- мы можем тратить средства позже, а получать прибыль раньше, благодаря этому с учетом коэффициентов прибыли нам придется терять меньше за счет процентной ставки на деньгах, которые мы тратим, и получать больше за счет процентной ставки на деньгах, которые мы получаем;
- мы можем увеличить вероятность того, что проект выживет. Таким образом мы с большей долей вероятности получим дополнительные выплаты в процессе дальнейшей работы над проектом.

Варианты

Есть еще один взгляд на экономику программного проекта — это набор вариантов, в соответствии с которыми может развиваться этот проект. Управление программным проектом может осуществляться в соответствии с одним из вариантов.

- Вариант закрыть проект — вы можете получить какую-либо прибыль даже в случае, если работа над проектом будет прервана. Чем большую выгоду вы получите от проекта, который был завершен, не достигнув изначально планировавшегося состояния, тем лучше.
- Вариант смены направления — вы можете изменить направление развития проекта. Стратегия управления проектом будет более выгодной в случае, если в процессе работы над проектом заказчики будут обладать возможностью сменить изначально сформулированные ими требования. Чем чаще это может происходить и чем кардинальнее заказчики могут менять свои требования, тем лучше.
- Вариант отсрочки решения — прежде чем инвестировать средства и ресурсы в том или ином направлении, вы можете подождать до тех пор, пока ситуация сама собой не разрешится и не станет более понятной для вас. Стратегия управления проектом будет более выгодной в случае, если вы будете обладать возможностью подождать с вложением денег, и при этом не потерять полностью возможность инвестирования средств. Чем более длительной может быть отсрочка и чем большее количество денег можно удержать от преждевременного инвестирования, тем лучше.
- Вариант роста инвестиций — если вы видите, что рынок начинает расти, вы можете оперативно увеличить инвестиции для того,

чтобы с выгодой воспользоваться этим. Стратегия управления проектом будет более выгодной в случае, если вы будете обладать возможностью все больше и больше расширять производство за счет увеличения объема инвестиций. Чем быстрее и чем дольше проект может расти, тем лучше.

Определение значимости каждого из вариантов — это на две части искусство, на пять частей — математика и на одну часть — старое доброе перетягивание каната.

При этом необходимо учитывать следующие пять факторов:

- объем инвестиций, необходимых для реализации того или иного варианта;
- цена, в рамках которой вы сможете достигнуть цели, если вы будете действовать в рамках того или иного варианта;
- текущая ценность поставленной вами цели;
- время, которое потребуется вам для того, чтобы реализовать тот или иной вариант;
- неопределенность, с которой вы можете оценить ценность поставленной вами цели.

Из всех этих факторов доминирующим как правило оказывается последний: неопределенность. На основании этого мы можем прийти к некоторым выводам и прогнозам относительно разрабатываемой нами стратегии. Представьте, что мы создали стратегию управления проектом, которая максимизирует полезность проекта на основе анализа вариантов и обеспечивает:

- частый возврат точных сведений о текущем состоянии разработки;
- множество возможностей в значительной степени изменить набор изначально заданных требований и направление развития проекта;
- меньший объем изначальных инвестиций;
- возможность при желании увеличить темпы разработки.

Чем большей будет неопределенность, тем полезнее окажется созданная нами стратегия. Это утверждение является справедливым вне зависимости от того, является ли причиной неопределенности технический риск, изменяющиеся условия, в которых функционирует бизнес, или быстро эволюционирующие требования заказчика. (Это является теоретическим ответом на вопрос: «Надо ли мне использовать XP?» Использовать XP следует в условиях, когда требования заказчика неопределенны или быстро меняются.)

Пример

Представьте, что вы занимаетесь программированием фактически в одиночку. Вы видите, что добавление в программу некоторой возможности обойдется вам в $10. Вы ожидаете, что вы сможете заработать на этой возможности приблизительно $15. Таким образом, чистая текущая ценность (Net Present Value, NPV) добавления в программу данной возможности составит $5.

Представьте, что вы не можете сказать точно, какова будет на самом деле ценность рассматриваемой вами возможности, — вы можете лишь предположить, что заказчик будет готов заплатить за нее $15. В действительности этот параметр может отличаться от предполагаемого вами значения на 100% в обе стороны. Теперь предположим, что если вы соберетесь добавлять данную возможность спустя год от текущего момента, то это все равно будет стоить вам те же $10 (см. главу 5).

Какова будет ценность стратегии, в рамках которой вы не будете реализовывать эту возможность прямо сейчас, а подождете в течение года? В настоящее время средняя процентная ставка составляет около 5% годовых. С учетом этой процентной ставки искомая ценность составит около $7,87.

Следовательно, стратегия годичного ожидания, прежде чем добавить в программу новую возможность, обойдется нам *дороже,* чем если бы мы, ничего не ожидая, *прямо сейчас* инвестировали деньги в разработку данной возможности (напомню, что на текущий момент соответствующая NVP составляет $5). Почему? В настоящее время мы находимся в неопределенности и не можем точно сказать, будет ли данная возможность действительно полезна для нашего заказчика и сможет ли он прямо сейчас начать зарабатывать на ней деньги. Если мы реализуем возможность прямо сейчас и возможность окажется действительно полезной, то наш заказчик через год получит за счет этого определенную прибыль. Однако может оказаться, что для нашего заказчика эта возможность не представляет никакой ценности, и поэтому, отказавшись на текущий момент от ее реализации, мы можем сэкономить собственные ресурсы.

Говоря проще, варианты помогают нам избавиться от нежелательного риска.

Четыре переменные

В наших проектах мы пытаемся контролировать четыре переменные — затраты, время, качество и объем работ. Из всех этих переменных наиболее удобной для контроля является объем работ.

В данной главе я расскажу вам о модели разработки программного обеспечения, которая представляет собой систему контролируемых переменных. В рамках дайной модели разработка программного обеспечения определяется с использованием следующих четырех переменных:

- затраты (cost);
- время (time);
- качество (quality);
- объем работ (scope).

В данном случае игра в разработку программного обеспечения выглядит следующим образом: внешние силы (заказчики, менеджеры) должны определить значения для любых трех переменных из указанного набора, при этом команда разработчиков должна выбрать результирующее значение для оставшейся переменной.

Некоторые менеджеры и заказчики полагают, что они обладают правом с успехом установить значение для всех четырех переменных. «Вы обязаны реализовать все, что указано в техническом задании к первому числу следующего месяца, работая в текущем составе, то есть без увеличения численности, при этом качество должно стоять на первом месте и не уступать нашим обычным стандартам». Когда происходит подобное, качество, как правило, летит ко всем чертям (и это, к сожалению, как раз и является общераспространенным стандартом), потому что никто не в со-

стоянии хорошо делать свою работу под слишком большим давлением. Помимо качества, время, как правило, также выходит из-под контроля. Таким образом, вы производите некачественное программное обеспечение, не успевая при этом сдать работу к сроку.

Чтобы решить проблему, необходимо сделать все четыре переменные наблюдаемыми. Если все — программисты, заказчики и менеджеры — смогут наблюдать за поведением всех четырех переменных, будет легче сознательно выбрать, какие из четырех переменных следует контролировать. Если результирующее значение четвертой переменной окажется неприемлемым, можно будет либо изменить входные значения, либо выбрать для контроля другие три переменные.

Взаимосвязь между переменными

Затраты (cost) — чем больше денег, тем легче работать, однако слишком большое количество денег в самые кратчайшие сроки создаст больше проблем, чем требуется решить. С другой стороны, если вы выделяете на проект слишком мало денег, вы не сможете решить поставленные перед вами заказчиком проблемы.

Время (time) — с увеличением объема времени, выделяемого на выполнение проекта, у вас появляется возможность повысить качество разрабатываемой программы, а также расширить объем работ. Однако отзывы о системе, которая уже эксплуатируется в реальных рабочих условиях, гораздо ценнее, чем любые другие отзывы, поэтому, если вы выделите для выполнения проекта слишком много времени, проект может пострадать. Если вы выделите для проекта слишком маленькое время, пострадает качество.

Качество (quality) — эта переменная контролируется хуже всего. Если вам необходимо достигнуть каких-либо краткосрочных целей (для достижения которых требуется несколько дней или недель), вы можете намеренно пожертвовать качеством, однако связанные с этим затраты — человеческие, деловые и технические — могут оказаться чрезмерными.

Объем работ (scope) — сократив объем работ, вы можете повысить качество (при условии, конечно, что поставленная заказчиком задача решена). Сокращение объема работ позволяет также сократить время проекта и связанные с проектом затраты.

Между рассмотренными четырьмя переменными не существует простой зависимости. Например, вы не сможете увеличить скорость работы программы, просто затратив на ее разработку больше денег. Как говорит народная мудрость, «девять женщин не могут родить ребенка за один месяц». (И в противоположность тому, что я слышал от некоторых

менеджеров, даже восемнадцать женщин не смогут родить ребенка за один месяц.)

В определенном смысле затраты являются наиболее ограничивающей переменной. Вы не можете напрямую менять деньги на качество, объем работ или скорость, с которой выпускаются промежуточные версии продукта. На самом деле в начале проекта вообще не существует возможности потратить сразу много денег. В начале работ инвестирование необходимо выполнять понемногу и затем, с течением времени, увеличивать объемы вложений. В процессе того, как проект будет развиваться, вы сможете эффективно тратить все большее и большее количество денег.

У меня был один клиент, который сказал мне: «Необходимо обеспечить всю заданную функциональность. Для этого предполагается использовать 40 программистов».

Я ответил: «В самый первый день работы над проектом вы не сможете задействовать сразу всех 40 программистов. Для начала необходимо задействовать небольшую команду. Затем к проекту необходимо подключить еще одну команду. После этого вы сможете использовать четыре программистских рабочих группы. В течение двух лет у вас появится возможность подключить к работе всех 40 программистов, однако в самом начале сделать этого нельзя».

Он настаивал: «Вы не понимаете. Мы обязаны задействовать в проекте 40 программистов».

Я ответил: «Вы не сможете сделать этого».

Однако он не унимался: «Но мы обязаны».

Они не смогли. Я имею в виду, что они попытались сделать это. Они наняли 40 программистов, однако дела пошли не самым лучшим образом. Все эти программисты поувольнялись. Они наняли еще 40 программистов. Через четыре года они только-только приступили к рабочим испытаниям результатов своей работы. В рабочих условиях начал функционировать небольшой подпроект, однако перед этим они чуть было не закрылись.

Любые связанные с затратами ограничения могут привести менеджеров в бешенство. В особенности это справедливо, когда они занимаются планированием бюджета на год. В подобных ситуациях они настолько привыкли выводить все из предполагаемого объема затрат, что зачастую приходится сталкиваться с очень серьезными ошибками. На самом деле, управляя объемом затрат, нельзя контролировать все и вся, и это обстоятельство ни в коем случае нельзя упускать из виду.

Существует еще одна связанная с затратами проблема. Рост затрат может быть вызван желанием привлечь к себе внимание или повысить свой престиж. «Ну да, под моим началом проект, в котором задействовано 150 человек (важное сопение)». В результате проект может умереть по той

простой причине, что его руководитель просто захотел выглядеть внушительно. Почему-то ему показалось, что если он сумеет разработать ту же самую систему, наняв для этой цели всего 10 программистов и закончив работу за половину выделенного для этой цели времени, он не сможет произвести на окружающих должного впечатления.

С другой стороны, затраты тесно связаны с другими переменными. Если, действуя в допустимых пределах, вы увеличите объем инвестиций, вы можете расширить объем работ, или вы можете действовать с большей свободой и улучшить качество, или вы можете (до определенной степени) уменьшить время, необходимое для завершения работы.

Затратив дополнительные деньги, вы также снижаете трение — вы получаете более быстрые компьютеры, большее количество технических специалистов, более просторные и удобные офисы.

Ограничения, связанные с временем реализации проекта, как правило появляются извне — в качестве наиболее свежего примера можно вспомнить проекты, связанные с решением проблемы 2000 года. В качестве примеров внешних временных ограничений можно привести также конец года, начало следующего квартала, дата планируемого завершения работы старой системы, крупная торговая выставка. Зачастую время не поддается контролю со стороны менеджеров проекта — его контролируют заказчики.

Качество — это еще одна весьма странная переменная. Зачастую, настаивая на улучшении качества, мы можете завершить проект быстрее, чем запланировано. Или вы можете успеть сделать больше за заданный интервал времени. Именно это случилось со мной, когда я приступил к разработке тестов для программного модуля, работа над которым описывалась в главе 2. Как только я закончил работу над всеми тестами, я был настолько уверен в своем коде, что смог разработать код модуля существенно быстрее, без каких-либо липших сомнений и размышлений. Я смог подчистить мою систему с меньшим количеством усилий, в результате я существенно упростил дальнейшую разработку. Мне часто приходится наблюдать, как подобное происходит с целыми командами разработчиков. Как только они приступают к тестированию или как только они разрабатывают общие для всех стандарты кодирования, работа начинает идти существенно быстрее.

Существует весьма странная зависимость между внутренним и внешним качеством. Внешнее качество — это качество, измерением которого занимается заказчик. Внутреннее качество оценивается программистами. Если вы намерены временно пожертвовать внутренним качеством для того, чтобы сократить время разработки, и при этом надеетесь на то, что внешнее качество не пострадает слишком сильно, имейте в виду, что вы стремитесь к достижению краткосрочной цели. Возможно, закрыв глаза на

качество внутренней отделки, вам удастся сэкономить пару недель или даже месяц, однако с течением времени количество внутренних проблем может увеличиться настолько, что разрабатываемую вами систему будет чрезвычайно сложно сопровождать и развивать; кроме того, возможно, вам не удастся достичь приемлемого уровня внешнего качества.

С другой стороны, время от времени вы попадаете в ситуацию, в которой вы можете завершить работу быстрее, ослабив ограничения, связанные с качеством. Однажды я разрабатывал систему, которая предназначалась для замены устаревшей системы, написанной на COBOL. Поставленный перед нами критерий качества подразумевал, что в рамках новой системы мы обязаны были в точности воспроизвести все ответы старой системы. По мере того как дата сдачи работы становилась все ближе и ближе, мы начали понимать, что можем воспроизвести в нашем новом программном продукте не только полную функциональность старой системы, но также и все заложенные в ней ошибки, однако для этого придется перенести сроки сдачи. Мы обратились к заказчикам и продемонстрировали им, что предлагаемые нами ответы являются более корректными. Мы также сообщили, что если в этом отношении они нам поверят, мы сможем сдать работу в срок.

С качеством связан значительный человеческий фактор. Каждый желает делать свою работу хорошо и каждый работает существенно лучше, если чувствует, что он делает свою работу хорошо. Если же вы намеренно жертвуете качеством, возможно, в первое время ваша команда действительно будет работать быстрее, однако в скором времени вступит в действие деморализация. На людей начнет давить ощущение, что они заняты производством бракованного продукта. Если вы откажетесь от тестирования или от пересмотра кода или от соответствия стандартам, возможно, на некоторое время вы добьетесь некоторого преимущества, однако в дальнейшем вы можете потерять его за счет человеческого фактора.

Фокус на объеме работ

Множество людей хорошо знает, что такое затраты, качество и время, и как с их помощью контролируется процесс производства программного обеспечения, однако при этом многие из них не признают четвертую переменную — объем работ. Объем работ, связанный с производством программного продукта, — это наиболее важная переменная, с которой приходится иметь дело в производстве программного продукта. В большинстве случаев как программисты, так и бизнесмены обладают весьма туманными представлениями о том, что является наиболее ценным в разрабатываемом программном продукте. Однако если вы активно управляете показателем объема работ, вы можете предоставить менеджерам и заказчикам контроль над затратами, качеством и временем.

Отличительной чертой этого показателя является то, что объем работ — сильно изменяющаяся переменная. В течение десятилетий программисты привыкли жаловаться: «Заказчики не могут сказать нам, чего они хотят. Когда мы даем им то, о чем они нас просили, они говорят, что им это не нравится». И это абсолютная горькая правда всей индустрии производства программного обеспечения. В самом начале работы над проектом требования заказчика никогда не бывают четкими и ясными. Заказчики никогда не могут сказать вам, что именно они хотят.

После разработки некоторого фрагмента программного продукта требования к нему изменяются. Как только заказчики видят первую версию продукта, они понимают, что они хотят увидеть во второй версии... или что они на самом деле хотели увидеть в первой версии. И это достаточно важный процесс познания, так как во многих случаях заказчики просто не могут сформулировать требования к продукту, которого в реальности пока что не существует. Зачастую не возможно четко и полностью сформулировать техническое задание на основе только лишь беспочвенных абстрактных размышлений и предположений. Многие важные умозаключения и выводы делаются заказчиками на основе опыта. Однако заказчики не имеют возможности получить свой опыт в одиночку. Они нуждаются в людах, которые умеют программировать. И эти люди должны быть не просто советчиками, они должны быть компаньонами.

Что, если посмотреть на нечеткость требований не как на проблему, а как на удобную возможность? В этом случае мы можем обнаружить, что показатель объема работ является наиболее удобной в управлении переменной из тех четырех переменных, о которых я говорил ранее. Так как этот показатель наименее четко очерчен, мы можем формировать его в соответствии с нашими собственными предпочтениями и предпочтениями заказчика — немного в эту сторону, немного в ту сторону. Если время поджимает и надо выпускать очередную версию продукта, у нас всегда будет что-то, что мы можем отложить до следующей версии. Однако если мы будем стараться не втискивать в рамки одной версии слишком много работы, то сохраним возможность выпустить продукт требуемого качества в указанные сроки.

Если мы создадим дисциплину разработки программного обеспечения на основе описанной модели, то получим возможность контролировать дату выпуска, качество и стоимость любого фрагмента программного продукта. Вначале мы можем рассматривать объем работ как функцию от первых трех переменных, однако в дальнейшем мы получаем возможность постоянно корректировать объем работ с учетом складывающихся условий.

Разрабатываемая нами дисциплина должна подразумевать низкую стоимость внесения изменений в проект, так как предполагается, что направ-

ление развития проекта будет меняться достаточно часто. Не стоит тратить деньги на разработку программного продукта, который в дальнейшем не будет использоваться. Не стоит тратить деньги на асфальтирование дороги, по которой вы все равно не будете ездить, так как вам придется свернуть на ближайшем повороте. Чтобы в подобной ситуации добиться желаемых результатов, вы должны использовать методику, в рамках которой стоимость внесения в проект изменений будет приемлемой в течение всего времени жизни системы.

Если вы будете отказываться от реализации важной функциональности в конце каждого очередного периода работы над очередной версией продукта, заказчик в скором времени разочаруется в вашей работе. Чтобы избежать этого, XP использует две стратегии.

1. В процессе работы, приступая к реализации очередной части проекта, вы постоянно делаете предположение относительно того, сколько времени и усилий вам потребуется для того, чтобы реализовать эту часть проекта. В дальнейшем вы сравниваете реально затраченное время с вашими предварительными оценками. Таким образом вы тренируете свою способность правильно оценивать свои силы и снижаете вероятность отказа от реализации ранее запланированной функциональности.
2. В первую очередь вы реализуете те требования заказчика, которые являются наиболее важными для него. Таким образом, если на завершающем этапе работы над очередной версией системы приходится отказываться от реализации какой-либо функциональности, эта функциональность оказывается менее важной, чем та функциональность, которая уже присутствует в системе.

Стоимость внесения изменений

При определенных условиях экспоненциально растущую относительно времени стоимость внесения изменений в систему можно сгладить. Если кривая роста стоимости внесения изменений в систему сглаживается, старые правила, определяющие наилучшие методики разработки программного обеспечения, перестают быть верными.

Одно из общепринятых фундаментальных правил, определяющих традиционную стратегию разработки программного обеспечения, утверждает, что по мере работы над проектом стоимость внесения изменений в разрабатываемый программный продукт увеличивается экспоненциально. Я помню, как, будучи студентом колледжа, сидел в огромной аудитории, пол которой был покрыт блестящим линолеумом, и смотрел, как профессор рисует на доске кривую, изображенную на рис. 1.

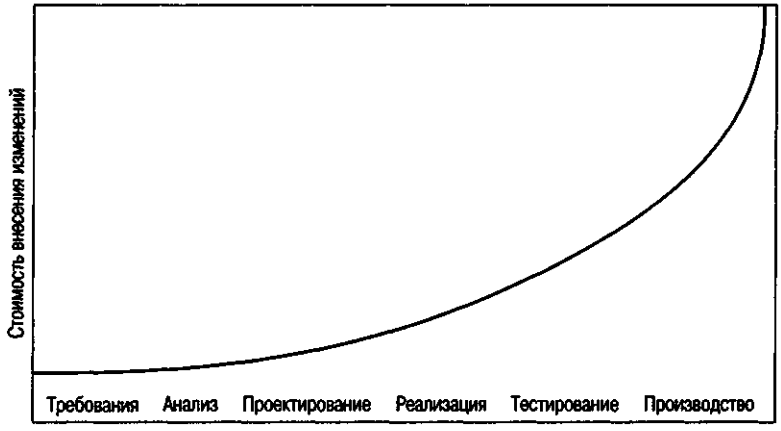


Рис. 1. С течением времени стоимость внесения изменений в программный продукт возрастает экспоненциально

«Затраты, связанные с исправлением проблемы, обнаруженной внутри программного продукта, растут экспоненциально с течением времени, — говорил профессор. — Проблема, для решения которой в процессе анализа требований потребовался бы доллар, может стоить вам нескольких тысяч долларов, если вы обнаружите ее в момент, когда система уже будет в производстве».

Тогда, в той аудитории я раз и навсегда решил для себя, что я никогда не позволю проблемам оставаться в разрабатываемых мною программах до самого производства. Ни за что! Я обязательно буду устранять проблемы так быстро, как только это возможно. Я буду бороться с проблемами, которые даже еще не возникли, а только могут возникнуть. Я буду постоянно и беспрерывно пересматривать и проверять мой код. Ни под каким видом я не позволю себе допустить ошибку, которая в будущем будет стоить моему нанимателю сотни тысяч долларов.

Однако проблема состоит в том, что на самом деле кривая, изображенная на рис. 1, больше не соответствует действительности. Говоря точнее, благодаря использованию новых технологий и специальных методик программирования можно добиться того, что кривая стоимости внесения изменений в программный продукт будет выглядеть прямо противоположно. Теперь становятся возможными истории, подобные следующей, которая случилась со мной относительно недавно, во время работы над системой управления контрактами страхования жизни:

17.00 — я обнаруживаю, что некоторая встроенная в нашу систему возможность — способность в рамках одной трансакции хранить несколько дебетов с нескольких учетных записей и несколько кредитов для нескольких учетных записей — в процессе функционирования системы, по сути дела, не используется. Каждая трансакция извлекает средства только с одной учетной записи и переносит их только на одну учетную запись. У меня в голове возникает вопрос: можно ли упростить систему, как показано на рис. 2?

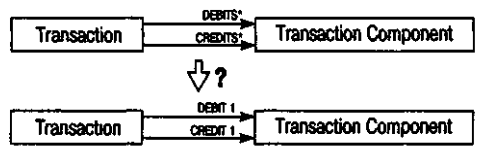


Рис. 2. Можем ли мы использовать только один компонент для каждой пары дебет/кредит?

17.02 — я обращаюсь к Массимо и прошу его, чтобы он подошел ко мне и сел рядом для того, чтобы помочь мне разобраться в ситуации. Мы пишем запрос. Выясняется, что из всех 300 000 трансакций в системе ни

одна не использует более одной учетной записи для снятия денег и более одной учетной записи для переноса денег. Каждой из трансакций соответствует только одна дебетная учетная запись и только одна кредитная учетная запись.

17.05 — если мы хотим исправить систему, как мы можем этого достичь? Мы можем изменить интерфейс компонента Transaction, а также изменить реализацию. Мы переписываем четыре требуемых метода и приступаем к тестированию.

17.15 — все тесты (более чем 1000 модульных и функциональных тестов) по-прежнему выполняются на 100% отлично. Мы не можем придумать ситуацию, в которой внесенные нами изменения могут стать причиной неработоспособности системы. Мы приступаем к работе над кодом миграции базы данных.

17.20 — мы завершаем программирование процедур миграции, которая должна быть выполнена в течение ночи. Также мы выполняем резервное копирование базы данных. После этого мы устанавливаем новую версию кода и инициируем процесс миграции.

17.30 — мы запускам несколько проверочных тестов. Все, что мы могли предусмотреть, работает великолепно. Не в состоянии придумать что-либо важное, о чем мы могли позабыть, мы уходим домой.

Следующий день — журнал ошибок пуст. Никаких жалоб от пользователей не поступает. По всей видимости, внесенные нами изменения отлично работают.

В течение нескольких следующих недель мы обнаружили, что в связи с использованием новой структуры разрабатываемую нами систему можно упростить еще больше. При этом мы получаем возможность внести в бухгалтерский раздел нашей программы совершенно новую удобную функциональность. Таким образом мы делаем систему проще, чище и менее избыточной.

В течение последних нескольких десятилетий сообщество людей, работающих в области разработки программного обеспечения, приложило огромные усилия для того, чтобы снизить стоимость внесения изменений. Для этой цели были разработаны более совершенные языки программирования, более совершенные технологии баз данных, более эффективные методики кодирования, улучшенные среды и инструменты разработки, новые правила.

Что, если все эти инвестиции не пропали даром? Что, если вся эта суета с новыми языками программирования и новыми базами данных не была напрасной? Что, если теперь стоимость внесения изменений не растет экспоненциально, как раньше, а увеличивается более медленно, со временем приближаясь к асимптоте? Что, если завтра профессор, преподаю-

щий студентам основы разработки программного обеспечения, нарисует на доске линию, изображенную на рис. 3?

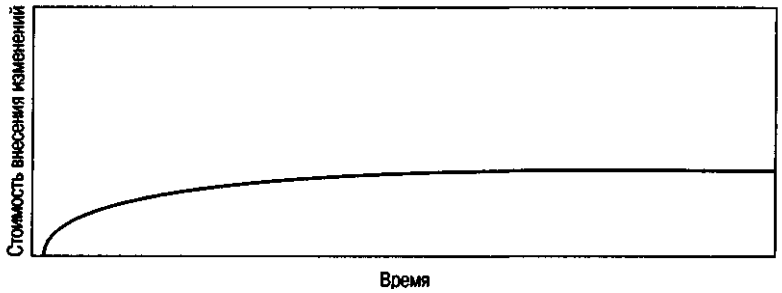


Рис. 3. Стоимость изменений со временем может увеличиваться существенно медленнее, чем экспонента

Именно это является одним из основных предположений XP. Это техническое предположение XP. Если стоимость внесения в систему изменений со временем растет достаточно медленно, стратегия разработки программы должна быть совершенно другой, отличной от той, которая используется в случае, если стоимость внесения в систему изменений со временем растет экспоненциально. В подобной ситуации вы можете откладывать решение важных задач на более поздние сроки. Вы получаете возможность принимать важные решения настолько поздно, насколько это возможно. Это делается для того, чтобы осуществлять связанные с этим затраты как можно позже. Кроме того, если вы откладываете решение важных вопросов на более поздний срок, тем самым вы повышаете вероятность того, что выбранное вами решение окажется правильным. Другими словами, сегодня вы должны реализовать только то, без чего сегодня не обойтись, при этом вы можете рассчитывать на то, что проблемы, решение которых вы отложили до завтра, развеются сами собой, то есть перестанут быть актуальными. Вы можете добавлять в дизайн новые элементы только в случае, если эти новые элементы упрощают код или делают написание следующего фрагмента кода более простым.

Если пологая кривая роста затрат делает XP возможным, то экспоненциальная кривая роста затрат делает XP невозможным. Если изменение обойдется вам в кругленькую сумму, вы сойдете с ума, пытаясь предугадать, каким образом это изменение повлияет на работу системы. Если же изменение обходится вам дешево, вы всегда можете рискнуть и проверить, что будет, если вы тем или иным образом измените код, — позже вы всегда можете изменить систему так, как это будет лучше. В подобной ситуации, внеся изменения в систему, вы через короткое время получаете ответную информацию о том, насколько полезны данные изменения для системы.

Но как поддерживать невысокую стоимость изменений на протяжении всего времени работы над проектом? Сохранение низкой стоимости изменений — это не какой-то магический трюк, оно достигается не просто так, а в результате применения технологий и методик, которые позволяют сделать программный продукт податливым и легко модернизируемым.

С технологической точки зрения ключевой технологией, позволяющей добиться этого, являются объекты. Обмен сообщениями между объектами позволяет существенно расширить спектр возможностей по изменению разрабатываемой системы. Каждое сообщение становится потенциальной точкой для внесения в систему грядущих модификаций, модификаций, которые могут вносится в систему, не затрагивая при этом существующий код.

Объектные базы данных переносят эту гибкость в пространство постоянной памяти. Благодаря использованию объектной базы данных вы получаете возможность с легкостью переносить информацию об объектах из одного формата в другой, так как в объектных базах данных код соединен с данными, а не отделен от них, как это было в более ранних базах данных. Даже если вы не можете найти способ выполнить миграцию объектов, вы можете обеспечить в рамках одной системы сосуществование двух разных реализаций.

Я вовсе не хочу сказать, что вы обязаны использовать объекты для того, чтобы получить гибкость. Лично я познакомился с основами XP, когда наблюдал, как мой отец пишет на ассемблере управляющий код процесса реального времени. Он разработал стиль, который позволил ему постоянно обновлять дизайн его программ. Все же мой опыт подсказывает мне, что стоимость изменений увеличивается в большей степени в случае, если вы не используете объекты, чем в случае, если вы основываете свой проект на объектно-ориентированном подходе.

Я также не хочу сказать, что объекты — это все, что вам потребуется для снижения стоимости затрат. Я видел (и даже, по правде сказать, сам написал) немало кодов, заниматься изменением которых я не пожелал бы и врагу.

Чтобы упростить модификацию вашего кода даже спустя несколько лет после начала работы над проектом, вы должны учитывать следующие факторы:

- простой дизайн, в котором отсутствуют лишние элементы, — никаких идей, которые на текущий момент не используются, однако предположительно могут использоваться в будущем;
- автоматические тесты — благодаря им всегда с легкостью можно узнать о том, что в результате внесения в систему изменений ее поведение изменилось;

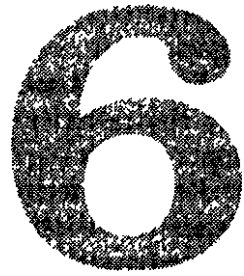
- постоянная практика в деле модификации дизайна системы — когда приходит время менять систему, вы не почувствуете страха перед этим.

Если мы заложим в основу нашей работы три перечисленных элемента — простой дизайн, автоматические тесты и опыт постоянного видоизменения системы, мы увидим, что кривая расходов, связанных с внесением в систему изменений, станет пологой, как на рис. 3. Изменение, для осуществления которого до начала кодирования потребовалось бы несколько минут, через два года работы над системой потребует от вас не более 30 минут. Однако я сталкивался с проектами, в которых на принятие и предварительное обдумывание подобных решений тратятся дни и даже недели, хотя вместо этого проблему можно решить уже сейчас, а завтра, если это потребуется, можно будет с легкостью все переделать на новый лад.

Теперь, когда мы кардинально пересмотрели наши изначальные предположения относительно стоимости внесения в систему изменений, у нас появляется возможность применить совершенно иной подход к разработке программного обеспечения. Как и любые другие подходы, разрабатываемый нами подход строг, он не терпит нарушения определяемых в его рамках правил, однако его строгость лежит в несколько иных измерениях. Вместо того чтобы заботиться о принятии важных решений как можно раньше и менее значительных решений — позже, мы с вами планируем разработать подход, в рамках которого каждое решение принимается быстро, при этом оно надежно защищается с использованием автоматических тестов. Помимо этого, в рамках нового подхода вы должны быть готовыми к изменению дизайна системы в случае, если вы понимаете, что существует более удачный дизайн, чем тот, который реализован вами на текущий момент.

Создание такого подхода — далеко не простая задача. Мы должны будем заново проанализировать и пересмотреть принятые нами ранее предположения о том, какая методика разработки программного обеспечения является хорошей, а какая — нет. Мы будем двигаться в сторону цели поэтапно. А начнем мы с рассказа, который будет притягивать к себе все остальное, чем мы будем заниматься.

Обучение управлению автомобилем



Управляя разработкой программного продукта, мы должны совершать множество незначительных преобразований и воздействий. Мы должны избегать применения нескольких существенных модернизаций. Другими словами, управление разработкой программной системы в некотором смысле должно напоминать управление едущим автомобилем. Это означает, что у нас должна быть обратная связь для того, чтобы вовремя узнать о том, что мы несколько отклонились от желаемого направления движения. Мы должны обладать самыми широкими возможностями по внесению в проект изменений, кроме того, мы должны обладать возможностью вносить эти изменения ценой приемлемых для нас затрат.

Теперь у нас есть общая формулировка проблемы — угрожающе огромная цена риска и возможность управлять этим риском при помощи вариантов, кроме того, мы обладаем ресурсом, который потребуется нам для формирования решения: свобода вносить изменения позже, не увеличивая при этом связанные с этим затраты. Теперь мы должны сконцентрировать усилия на поиске решения. Самая первая вещь, которая нам для этого потребуется, — это метафора — образно оформленный рассказ, к которому мы могли бы обращаться время от времени в случае стресса или для поддержки.

Я отлично помню тот день, когда впервые начал учиться водить машину. Я и моя мать сели в машину и выехали на автостраду Interstate 5 вблизи Чико (Chico) в штате Калифорния. Это прямой как стрела пустынный участок шоссе, выходящий из-под колес машины и, подобно натянутой струне, устремляющийся вдаль — к линии горизонта. Моя мать позволила мне пересесть в водительское кресло, а сама села на место переднего пассажира. И мы поехали. В начале я с осторожностью изучил, как именно движение рулевого колеса влияет на направление движения автомобиля, затем, освоившись, я позволил себе несколько расслабиться. «Управлять машиной надо так, — учила меня моя мама, — направь машину

строго по середине дороги и езжай по этой дороге прямо в направлении горизонта».

Мы ехали достаточно долго, и где-то посередине нашего путешествия что-то отвлекло мое внимание. Машина стала смещаться в сторону края дороги. Моя мать несколько обеспокоилась...

Когда колеса зашуршали по гравию на обочине, я немедленно опомнился. Моя мама (сейчас ее тогдашнее хладнокровие кажется мне ошеломляющим) плавно поправила руль так, чтобы машина вернулась на центр дороги. После этого она дала мне наиболее важное наставление относительно того, как следует управлять автомобилем: «Чтобы управлять автомобилем, вовсе не обязательно добиваться от него, чтобы он постоянно ехал в жестко заданном направлении. Достаточно внимательно следить за тем, куда едет машина, и поправлять направление ее движения — чуть-чуть влево, затем чуть-чуть вправо».

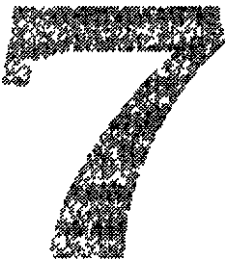
Это наставление является парадигмой для XP. В рамках XP не существует такой вещи, как прямое направление движения. Даже если вам кажется, что дела идут замечательно, вы не должны отрывать ваших глаз от дороги. Неизменными остаются только сами изменения. Всегда будьте готовы к тому, чтобы изменить направление чуть-чуть в одну сторону, затем чуть-чуть в другую сторону. В некоторых ситуациях вам придется менять направление так, что вы начнете движение в совершенно другую сторону. Такова жизнь программиста.

Абсолютно все в мире программного обеспечения меняется. Требования заказчиков меняются. Дизайн меняется. Бизнес видоизменяется. Технологии меняются. Команда разработчиков меняется. Члены команды разработчиков меняются. Сама по себе проблема остается неизменной, так как изменение рано или поздно должно произойти. На самом деле проблема состоит в том, что когда происходит изменение, люди не в состоянии с ним справиться.

Шофером программного проекта является заказчик. Если программа не делает того, чего заказчик от нее хочет, — это ваша неудача. Конечно же, заказчик не может сказать точно, чего именно он хочет от программы. Именно поэтому разработка программного продукта должна напоминать управление автомобилем. Редко когда кто-либо едет на машине, стараясь, чтобы автомобиль ни на миллиметр не отклонился от желаемой жестко заданной прямой. Обычно люди стараются обеспечить движение в заданном направлении, слегка подправляя направление движения при помощи рулевого колеса. Как программисты мы должны предоставить заказчику рулевое колесо, а также обеспечить обратную связь, как можно чаще сообщая ему, в каком именно месте дороги находится наш автомобиль.

Рассказ об управлении автомобилем также определяет мораль процесса XP. Описанные в следующей главе четыре ценности — коммуникация, простота, обратная связь и храбрость — дают представление о том, как должен выглядеть процесс разработки программного обеспечения. Однако методики, благодаря которым это достигается, будут отличаться для разных мест, разного времени и разных людей. Управляя процессом разработки, вы используете простой набор методик и стараетесь сформировать этот процесс так, как это описывается далее. По мере того как разработка продолжается, вы постоянно следите за тем, какие из методик требуют улучшения, а какие методики плохо соответствуют поставленной цели. Каждая методика — это эксперимент, который необходимо проводить до тех пор, пока не станет ясно, что он неадекватен.

Четыре ценности



Мы сможем успешно решить стоящую перед нами проблему, если сформулируем стиль, который направлен на прославление каждой из согласующегося набора ценностей, которые служат как **человеческим,** так и коммерческим требованиям: **коммуникация,** простота, обратная связь и храбрость.

Прежде чем на основе рассказа об управлении автомобилем сформировать набор методик разработки программного обеспечения, мы должны найти некоторый критерий, благодаря которому сможем узнать о том, что движемся в правильном направлении. Согласитесь, что было бы плохо, если бы мы создали стиль разработки, а потом узнали, что этот стиль нам не нравится или что он не срабатывает.

Краткосрочные индивидуальные цели часто конфликтуют с долгосрочными социальными целями. Общество решает эту проблему при помощи набора ценностей, подкрепленных мифами, ритуалами, наказаниями и наградами. Без уважения к этим ценностям люди забывают о социальных нуждах и стремятся реализовать свой собственный индивидуальный краткосрочный интерес.

Четыре ценности для XP — это:

- коммуникация (communication);
- простота (simplicity);
- обратная связь (feedback);
- храбрость (courage).

Коммуникация

Первая ценность XP — это коммуникация. Проблемы, которые возникают в процессе работы над проектом, почти всегда связаны с тем, что кто-то не сказал кому-то о чем-то важном. Иногда программист не сообщает

кому-то о важном изменении в дизайне. Иногда программист не задает заказчику важного вопроса, и в результате важное принятое им решение оказывается неправильным. Иногда менеджер не задает программисту важного вопроса, в результате у него складывается неполное, а иногда и неправильное представление о состоянии проекта.

Плохая коммуникация — это не случайность. Существует огромное количество предпосылок, которые ведут к нарушению коммуникации. Например, программист сообщает менеджеру плохие новости и менеджер наказывает программиста. Заказчик сообщает программисту нечто важное, а программист делает вид, что не понял или просто проигнорирует эту информацию.

Дисциплина XP нацелена на обеспечение непрерывной, постоянно осуществляемой коммуникации между участниками проекта. В рамках XP используются многие методики, которые невозможно реализовать без коммуникации. Эти методики направлены на достижение краткосрочных целей, например, тестирование модулей, программирование парами, а также оценка сложности задачи. В процессе тестирования, программирования парами и формирования предварительных оценок программисты, заказчики и менеджеры вынуждены тесно взаимодействовать.

Это не означает, что излишние разговоры мешают работе. Люди часто попадают в ситуацию, когда они сомневаются, делают ошибки, отвлекаются. В рамках XP существует специальное ответственное лицо — инструктор (coach), в чьи обязанности входит следить за тем, чтобы люди общались тогда, когда это надо. Если инструктор замечает, что люди перестают общаться, он стимулирует коммуникацию.

Простота

Вторая ценность XP — это простота. Инструктор XP спрашивает команду: «Какова самая простая вещь, которая скорее всего сработает?» (эта фраза является часто употребляемым выражением, в определенном смысле девизом, так что в мире XP даже существует специальная аббревиатура — Do The Simplest Thing That Could Possibly Work[1], DTSTTCPW).

Простота — это далеко не так просто. Напротив, это очень даже сложно — не обращать внимания на вещи, которые вы намерены реализовать завтра, на следующей неделе, в следующем месяце. Вы должны понимать, что если вы намеренно пытаетесь предугадать, как в будущем будет развиваться проект, это значит, что вы боитесь экспоненциального роста стоимости изменений. Программист боится, что завтра ему придется тратить массу усилий для исправления ошибок, которые он сдела-

[1] Сделай самую простую вещь, которая только сможет работать. — *Примеч. ред.*

ет сегодня, в результате он зачастую делает *код* более сложным. Время от времени инструктор должен мягко напоминать программистам, работающим над проектом, что вместо того, чтобы заниматься решением текущих задач, они пытаются прислушаться к собственным внутренним страхам. «Если ты пытаешься заставить работать это динамически сбалансированное бинарное дерево, значит, ты умнее, чем я. У меня складывается впечатление, что в данном случае можно обойтись обычным линейным поиском».

Грег Хатчинсон (Greg Hatchinson) пишет:

> Один человек, которого я консультировал, пришел к выводу, что для отображения текста нам потребуется универсальное диалоговое окно. Мы обсудили интерфейс этого диалога и как он будет работать. Программист решил, что диалог должен быть достаточно совершенным, он должен автоматически менять собственный размер и количество символов переноса строки в отображаемом тексте в зависимости от размера шрифта и других факторов. Я поинтересовался у него, какое количество других программистов, так же как он, нуждается в подобном диалоговом окне. Он ответил, что на текущий момент такое диалоговое окно нужно только ему. Я предложил не делать диалоговое окно столь интеллектуальным и ограничиться более узким набором возможностей. В этом случае диалоговое окно можно было бы реализовать минут за двадцать. Мы могли бы сделать класс с известным интерфейсом, а затем, когда появится такая необходимость, мы всегда могли бы усовершенствовать наше окно так, как этого потребует конкретная ситуация. К сожалению, я не смог убедить его, *в* результате он потратил два дня на реализацию своей идеи. На третий день условия задачи, для решения которой требовалось данное диалоговое окно, изменились и надобность в подобном диалоговом окне полностью отпала. Таким образом, два человеко-дня ушло на то, чтобы решить проблему, которая сразу же после этого перестала быть актуальной. Все же если вы сможете найти применение для данного кода, сообщите мне об этом. (Источник: электронная почта.)

XP делает свою ставку. Предполагается, что лучше сделать простую вещь сегодня и заплатить чуточку больше завтра для того, чтобы модифицировать ее (если в этом возникнет необходимость), чем разработать более сложный код сегодня, а потом узнать, что этот код больше не нужен.

Простота и коммуникация обладают замечательной взаимоподдерживающей связью. Чем больше вы общаетесь, тем яснее для вас становится, что именно вы должны сделать, и тем больше вы уверены в том, чего делать не надо. Чем проще система, тем меньше тем для обсуждения, а значит, тем полнее становится общение, особенно если вы упрощаете систему настолько, что для работы над ней требуется меньшее количество программистов.

Обратная связь

Третья ценность в XP — это обратная связь. Инструктор XP часто произносит: «Не спрашивай у меня, спроси у системы» и «Ты еще не написал

для этого тестовый случай?» Обратная связь, обеспечивающая точные и конкретные данные о текущем состоянии системы, — это воистину бесценная вещь. Оптимизм — это профессиональная болезнь всего программирования. Обратная связь — это лекарство от этой болезни.

Обратная связь работает в разных временных масштабах. Во-первых, обратная связь работает в масштабе минут и дней. Программисты пишут тесты для всей логики в системе. Любой из этих тестов может не сработать. Так программист получает обратную связь, которая ежеминутно обеспечивает его сведениями о состоянии системы. Когда заказчик пишет новые «истории» (описания возможностей системы), программисты немедленно оценивают их, благодаря чему заказчик получает обратную связь, которая обеспечивает его сведениями о качестве его историй. Человек, который следит за своевременным решением задач в рамках проекта, обеспечивает всех членов команды сведениями о том, какова вероятность того, что запланированный объем работ будет реализован в установленные сроки.

Обратная связь также работает в масштабе недель и месяцев. Заказчики и те, кто обеспечивает функциональное тестирование системы, разрабатывают функциональные тесты для всех историй (говоря иначе — упрощенных тестовых случаев), реализованных в рамках системы. Таким образом они обладают обратной связью, которая обеспечивает их информацией о текущем состоянии используемой ими системы. Заказчики пересматривают график работ каждые две или три недели для того, чтобы убедиться, что скорость выполнения работ соответствует плану. В случае необходимости план пересматривается. Эксплуатация системы в реальных производственных условиях начинается, как только система становится способной решать хотя бы небольшую часть возлагаемых на нее обязанностей. В результате бизнес получает возможность на практике оценить, как именно выглядит система и в каком направлении ее лучше всего развивать в дальнейшем.

О том, что к эксплуатации системы следует приступать как можно раньше, следует рассказать подробнее. Одной из стратегий в процессе планирования является правило, в соответствии с которым наиболее полезные для заказчика истории реализуются программистами и начинают эксплуатироваться как можно раньше. Благодаря этому программисты получают обратную связь, которая обеспечивает их сведениями о качестве принятых ими решений. Процесс разработки начинает напоминать управление автомобилем — программисты прилагают усилия для того, чтобы проект развивался в направлении, удобном для заказчика, при этом колеса должны всегда оставаться на асфальте. Некоторые программисты занимаются разработкой системы длительное время до того, как она начнет эксплуа-

тироваться в реальных рабочих условиях. Откуда же они могут узнать о том, что выполняемая ими работа действительно качественна и необходима?

В большинстве проектов используется прямо противоположная стратегия. Многие рассуждают так: «Как только система начинает использоваться на реальном производстве, в нее нельзя будет внести „интересных" изменений, поэтому система должна находится в стадии разработки настолько долго, насколько это возможно».

В XP делается прямо противоположное. «В разработке» — это временное состояние, в котором система находится в течение очень небольшого времени своей жизни. Будет значительно лучше, если система будет жить самостоятельной жизнью независимо от разработки. Необходимо позволить ей «дышать» и действовать самостоятельно. Необходимо поддерживать функционирование системы в условиях реального производства и одновременно с этим, разрабатывать новую функциональность. Необходимо обеспечить параллельную разработку и эксплуатацию, и чем раньше вы этого добьетесь, тем лучше.

Обратная связь работает совместно с коммуникацией и простотой. Чем более исчерпывающей является обратная связь, тем легче осуществлять коммуникацию. Когда кто-то недоволен написанным вами кодом и передает вам тестовый случай, который указывает на ошибку, это заменяет вам тысячу часов пространных дискуссий на тему эстетики программного дизайна. Если вы хорошо осуществляете коммуникацию, вы лучше знаете, что следует тестировать в системе. Простые системы тестировать проще. Разработка теста концентрирует ваше внимание на том, насколько простой может быть система; до тех пор, пока тест не сработает, вы не можете считать работу завершенной, а когда срабатывают все тесты, можно считать, что вы решили поставленную перед вами задачу.

Храбрость

В контексте первых трех рассмотренных ранее ценностей — коммуникации, простоты и обратной связи — необходимо действовать с максимально возможной скоростью. Если вы работаете со скоростью, которая не является абсолютно максимальной, это будет делать вместо вас кто-то другой, в результате этот кто-то прибежит к финишу первым и съест ваш обед вместо вас.

Я расскажу вам о том, как храбрость срабатывает в реальной жизни. В середине восьмой итерации включающего в себя 10 итераций рабочего графика (25 из 30 недель) первой версии первого крупного XP-проекта команда обнаружила фундаментальную ошибку в архитектуре системы.

Поначалу функциональные тесты указывали на хорошее качество разрабатываемой системы, однако позже количество набранных нами очков резко снизилось. В результате исправления одного дефекта обнаруживался другой дефект. Количество дефектов увеличивалось. Проблема была в архитектурном изъяне.

Для любопытных скажу, что мы работали над системой начисления выплат. Для хранения долгов компании перед сотрудниками использовались поля данных с названием Entitlement (жалованье), а для хранения долгов сотрудников перед другими людьми использовались поля с названием Deduction (вычет). Для некоторых людей использовалось отрицательное жалованье, в то время как вместо этого надо было использовать положительный вычет.

Команда поступила так, как должна была поступить. Когда все поняли, что путь вперед закрыт, они исправили архитектурный изъян. При этом половина всех используемых в отношении системы тестов перестала срабатывать. Однако в течение нескольких дней напряженных усилий по исправлению ситуации тесты снова начали срабатывать и качество системы, оцениваемое при помощи функциональных тестов, повысилось. Однако для того, чтобы поступить описанным образом, потребовалась отвага.

Еще один смелый ход — отказ от ранее разработанного кода. Представьте, что в течение всего рабочего дня вы работаете над реализацией некоторой функциональности. Работа идет неплохо, но когда вы близки к ее завершению, компьютер зависает. На следующее утро вы приходите на работу и в течение получаса восстанавливаете то, над чем работали весь предыдущий день, однако на этот раз код получается более чистым и более простым.

Используйте это. Если приближается конец рабочего дня и код все еще не поддается контролю, выбросьте его. Может быть, следует сохранить тестовые случаи, если вам понравился разработанный вами интерфейс. Однако это не обязательно. Возможно, следующим утром будет легче начать с нуля.

Возможно, перед вами три варианта дизайна. Вы можете потратить по одному дню на реализацию каждой из альтернатив для того, чтобы почувствовать, как они будут вести себя на практике. Затем выбросьте код и начните с нуля развивать тот вариант дизайна, который показался вам наиболее многообещающим.

Стратегия проектирования в XP напоминает алгоритм взбирания на холм (hill climbing). Вы делаете простой дизайн, затем вы делаете его более сложным, далее вы его упрощаете, потом опять усложняете. Проблема подобных алгоритмов состоит в том, что вы ищете локальный опти-

мум, — при этом никакое незначительное изменение не может улучшить ситуацию, однако улучшения можно достичь, используя значительное изменение.

Достигнув локального оптимума вы, возможно, упускаете более эффективный вариант дизайна. Что поможет вам избежать этого? Храбрость. Как только у кого-нибудь из вашей команды возникает сумасшедшая идея, в результате реализации которой сложность всей системы существенно уменьшится, он обязательно попробует реализовать свою идею, если конечно, у него хватит храбрости. Иногда это срабатывает. Если у вас хватит храбрости, вы приступите к эксплуатации новой версии системы в промышленных условиях. Считайте, что теперь вы забираетесь на совершенно новый холм.

Если у вас нет остальных трех ценностей, храбрость сама по себе является обычным взломом (в самом уничижительном смысле этого слова). Однако в сочетании с коммуникацией, простотой и надежной обратной связью храбрость становится чрезвычайно полезной.

Коммуникация идет на пользу храбрости, так как благодаря коммуникации вы обретаете возможность для осуществления более рискованных и более заманчивых экспериментов. «Тебе это не нравится? Я просто ненавижу этот код! Давай вместе посмотрим, в какой мере мы сможем переделать его сегодня днем». Простота идет на пользу храбрости, так как, обладая более простой системой, вы сможете позволить себе более смелые действия в ее отношении. Маловероятно, что вы нарушите ее функционирование по неизвестным причинам. Храбрость способствует простоте, ведь как только вы видите способ упростить систему, вы немедленно пробуете его реализовать. Надежная обратная связь идет на пользу храбрости, так как в процессе серьезной модернизации кода вы чувствуете себя значительно увереннее, если вы можете щелкнуть на кнопке и увидеть, что в результате тестирования все тесты показывают зеленый цвет. Если зеленым цветом окрашиваются не все тесты, вы переделываете или просто выкидываете свой код.

Ценности на практике

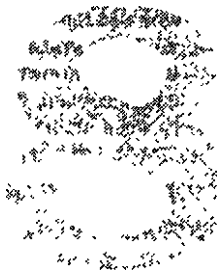
Я обратился к команде С3 (именно эти люди работали над тем самым крупным XP-проектом, о котором я говорил чуть раньше) с просьбой рассказать мне о моменте в процессе работы над проектом, который показался им наиболее достойным восхищения. Я надеялся услышать истории о переделке кода, о спасении благодаря тестам или по крайней мере о том удовлетворении, которое испытывал заказчик, используя разработанную систему на практике. Вместо этого я услышал следующее.

Для нас наиболее достойным был момент, когда Эдди сменил работу, чтобы ему было легче добираться до дома. Он ушел от нас на новое место, и в результате получил возможность экономить два часа ежедневно, благодаря чему он мог большее время проводить со своей семьей. Команда отнеслась к этому с уважением. Никто не сказал ему плохого слова. Каждый из нас просто поинтересовался, не может ли он чем-нибудь помочь.

Не указывает ли это на еще одну ценность? Ценность, которая лежит глубже, чем четыре рассмотренные нами ценности. Эта ценность есть уважение. Если члены команды не заботятся друг о друге и о том, чем они заняты, методика XP обречена. Скорее всего, это справедливо не только в отношении XP, но и в отношении других подходов к разработке программ (равно как и многим другим занятиям), однако XP наиболее чувствительна к этому. При должном сочувствии и интересе XP снижает трение между всеми рассмотренными элементами, обеспечивая их более гладкое взаимодействие. Если члены команды не заботятся о проекте, ничто не сможет спасти его. При минимальном сочувствии XP обеспечивает позитивную отдачу. Это не вопрос воздействия. Это своего рода удовольствие — быть частью чего-то, что работает, вместо того, чтобы быть частью чего-то, что не работает.

Все эти возвышенные разговоры — это, конечно, хорошо, однако если мы не сможем найти способ реализовать их на практике, если мы не сможем ввести их в действие для того, чтобы сделать рассмотренные ценности привычкой, все наши размышления и попытки окажутся всего лишь еще одной безрассудной попыткой преодолеть болото добрых методических намерений. Кроме того, нам необходимо получить более конкретное руководство, при помощи которого мы сможем разработать набор методик, удовлетворяющих четырем ценностям — коммуникации, простоте, обратной связи и храбрости — и воплощающих эти ценности в жизнь.

Базовые принципы



Исходя из четырех ценностей мы сформулируем десяток (или около того) принципов, в соответствии с которыми будет формироваться наш стиль. В дальнейшем мы будем проверять рассматриваемые методики на соответствие этим принципам.

Рассказ об управлении автомобилем учит нас делать множество небольших изменений и при этом не отрывать глаз от дороги. Четыре ценности — коммуникация, простота, обратная связь и храбрость — предоставляют нам критерий для проверки качества решения. Все же четыре ценности слишком туманны для того, чтобы в значительной степени помочь нам в оценке, какие именно методики мы должны использовать. Мы должны извлечь из описанных ценностей конкретные принципы, которые мы сможем использовать, оценивая ту или иную методику.

Эти принципы помогут нам выбирать между несколькими альтернативами. Предпочтение будет отдаваться альтернативе, которая соответствует принципам в большей степени. Каждый из принципов является воплощением ценностей, о которых я рассказал в предыдущей главе. Ценность может быть туманной: например, то, что для одного человека является простым, для другого человека является сложным. Принцип — это нечто более конкретное. Либо вы используете быструю обратную связь, либо нет. Вот перечень фундаментальных принципов:

- быстрая обратная связь;
- приемлемая простота;
- постепенное изменение;
- приемлемое изменение;
- качественная работа.

Быстрая обратная связь — психология обучения учит нас тому, что время между воздействием и ответной реакцией имеет огромное значение при обучении. Эксперименты с животными показывают, что даже незначительная разница в задержках ответной реакции влечет за собой существенные отличия в обучении. Задержка в несколько секунд между стимулом и ответом приводит к тому, что мышь не может понять, что красная кнопка означает еду. Таким образом, одним из принципов должна стать наиболее высокая быстрота, с которой сведения о состоянии системы передаются тем, кто в них заинтересован. В рамках нашей дисциплины необходимо обеспечить быстрое получение этих сведений, быструю интерпретацию и быстрое внесение в систему модификаций, сформированных на основе анализа этих сведений. Все это необходимо выполнять с максимально возможной скоростью. Бизнес должен быстро определять, каким образом система будет полезной для него, и он должен возвращать разработчикам эти сведения в течение дней или недель, но не в течение месяцев или лет. Программисты должны изучать, каким образом лучше всего проектировать, реализовывать и тестировать систему, необходимые для этого сведения должны поступать к ним в течение секунд или минут, но не дней, недель и месяцев.

Приемлемая простота — необходимо решать каждую проблему так, как если бы ее можно было решить самым смехотворно простым способом. Времени, которое вы экономите на решении 98% проблем, для которых это утверждение является истинным, вполне хватает, чтобы справиться с решением оставшихся 2% проблем. Во многих смыслах этот принцип является наименее привычным и наиболее трудным для программистов. В рамках установившихся традиций мы приучены к тщательному планированию своих действий, мы привыкли проектировать код с расчетом на его дальнейшее повторное использование. Однако в рамках XP огромные усилия (тестирование, переработка кода, коммуникация) прикладываются для того, чтобы сегодня программист думал о решении только сегодняшних проблем и был уверен в том, что завтра в случае необходимости имеющийся код можно будет с легкостью усовершенствовать так, как этого требует складывающаяся ситуация. Экономика разработки программ, представленная в виде набора нескольких вариантов, приветствует данный подход.

Постепенное изменение — объемные изменения, в рамках которых за один раз меняется абсолютно все, не срабатывают. Даже в Швейцарии, центре дотошного планирования, где я живу в настоящее время, люди избегают делать масштабные изменения. Любая проблема решается при помощи серии небольших изменений, в результате которых достигается желаемый эффект.

Как будет показано, постепенное изменение применяется в XP во многих областях и многими способами. Дизайн изменяется понемногу за один раз. План меняется понемногу за один раз. Команда меняется незначительно за один раз. Даже сама дисциплина XP должна внедрятся постепенно — маленькими шажками.

Приемлемое изменение — лучшей стратегией является та, которая решает наиболее актуальную для вас проблему и при этом сохраняет для вас максимальную свободу дальнейших действий.

Качественная работа — никто не любит плохо работать. Каждому нравится делать свою работу на «отлично». Из четырех ранее рассмотренных переменных (объем работ, затраты, время и качество) качество на самом деле не является свободно изменяемой переменной. Единственно возможными для нее значениями являются «превосходно» и «невероятно превосходно» — выбор между этими двумя значениями зависит от того, поставлены ли на карту человеческие жизни. В противном случае ваша работа вам не нравится, вы работаете плохо и ваш проект необратимо утекает в сточную канаву.

Далее приводится перечень менее важных принципов. Эти принципы все же могут помочь нам в определенных ситуациях:

- обучение обучению;
- небольшие изначальные инвестиции;
- игра для того, чтобы победить;
- надежное экспериментирование;
- открытая честная коммуникация;
- работа в соответствии с человеческими инстинктами, а не вопреки им;
- принимаемая ответственность;
- локальная адаптация;
- путешествие налегке;
- откровенные оценки.

Обучение обучению — вместо того чтобы сформировать набор доктрин наподобие «вы должны тестировать так-то и так-то», мы должны сконцентрироваться на стратегиях обучения, благодаря которым мы смогли бы научиться заранее определять, какой объем тестирования нам потребуется. Точно так же мы не должны жестко определять объем проектирования, переработки и любых других действий. Мы должны научиться определять необходимый объем по ходу дела. Некоторые идеи мы можем принять со всей уверенностью. В отношении других идей мы будем ме-

нее уверенными. В отношении таких идей читатели должны самостоятельно определить, нуждаются ли они в них или нет.

Небольшие изначальные инвестиции — если на слишком ранней стадии вы вложите в проект слишком много денег, вы приведете этот проект к краху. Стесненный бюджет принуждает программистов и заказчиков избегать слишком завышенных требований и использовать более осторожные подходы. Обладая скромным бюджетом, вы стараетесь извлекать максимальную прибыль из того, чем вы обладаете, и с максимальной пользой использовать имеющиеся ресурсы. Однако чрезмерный недостаток ресурсов — это тоже плохо. Если у вас не хватает ресурсов на решение даже всего одной интересной для вас проблемы, разрабатываемая вами система перестанет быть для вас интересной. Если над вами нависает некто, кто диктует вам объем работ, сроки окончания, уровень качества и затраты, вы вряд ли сможете управлять проектом так, чтобы привести его к желаемому успешному финалу. А вообще, как показывает практика, в большинстве случаев каждый может обойтись меньшим запасом ресурсов, чем тот запас, с которым он чувствует себя комфортно.

Игра для того, чтобы победить, — для меня всегда доставляет удовольствие смотреть, как играет баскетбольная команда UCLA Джона Вудена (John Wooden). Как правило, эти ребята выходят из поединка победителями. Однако даже тогда, когда игра близка к завершению, ребята из UCLA уверены, что они играют для того, чтобы победить. Конечно же, до этого они уже были победителями много-много раз. Они несколько расслаблены. Однако при этом они делают все для того, чтобы выиграть. И они выигрывают вновь.

Я вспоминаю игру баскетбольной команды из Орегона, которая была ярким контрастом. Орегон сражался с Аризоной, команда которой обладала национальной номинацией. Четыре игрока из Аризоны играли в национальной сборной NBA. Однако на половине матча неожиданно для всех Орегон оказался впереди на целых 12 очков. Игроки из Аризоны не могли ничего с этим поделать. Защита Орегона постоянно отражала их атаки. Однако после перерыва Орегон снизил темп игры и стал играть, экономя силы. Они закрывали глаза на медленно сокращающуюся разницу в счете, так как поставили перед собой задачу — просто удержать победу. И, конечно же, эта стратегия не сработала. Аризона немедленно воспользовалась своим преимуществом в опыте и выиграла игру.

Отличие состоит в том, что в одном случае команда играет для того, чтобы выиграть, а в другом случае — для того, чтобы не проиграть. Большинство игроков в мире разработки программного обеспечения, с которыми мне приходилось иметь дело, играют для того, чтобы не проиграть. Изводится масса бумаги. Проводится огромное количество совещаний.

Каждый пытается разрабатывать в строгом соответствии с общепринятыми правилами (по книжке) не потому, что в этом есть смысл, а потому, что в конце работы они получат возможность сказать, что это не их вина в том, что все кончилось так плачевно.

Если разработка программного продукта осуществляется для того, чтобы победить, каждый член команды делает все, чтобы помочь команде выиграть и не делает чего-либо такого, что может помешать этому.

Надежное экспериментирование — каждый раз, когда вы принимаете решение и не тестируете его, существует вероятность, что принятое вами решение неправильно. Чем больше таких решений вы принимаете, тем выше становится эта вероятность. Именно так увеличивается риск. Чтобы снизить риск, результатом сеанса проектирования должен стать не завершенный дизайн, а серия экспериментов, отвечающих на вопросы, которые возникли у вас в процессе проектирования. Результатом обсуждения требований также должна стать серия экспериментов. Каждое абстрактное решение должно быть протестировано.

Открытая честная коммуникация — это настолько очевидный принцип, что я чуть не забыл его упомянуть. Для всех очевидно, что общение должно быть открытым и искренним, однако зачастую приходится сталкиваться с обратным. И это огорчает. Программисты должны быть способны объяснить последствия решений, принимаемых другими людьми. Например: «В этом куске кода ты нарушил принцип инкапсуляции, и в результате у меня возникли проблемы». Программисты должны быть способны говорить друг другу о проблемах в коде. Они должны быть способны свободно и без стеснения рассказать о своих страхах и в ответ получить поддержку. Они должны быть способны с легкой душой сообщать заказчикам и менеджерам плохие новости. Они должны делать это как можно раньше, и их не надо за это наказывать.

Если я вижу, как кто-то, прежде чем ответить на мой вопрос, оглядывается вокруг, чтобы посмотреть, кто его слышит, я воспринимаю это как серьезную проблему всего проекта. Если обсуждаются личные вопросы, я, конечно же, понимаю необходимость некоторой конфиденциальности, однако вопрос о том, какую из двух объектных моделей использовать, не должен быть тайной за семью печатями.

Работа в соответствии с человеческими инстинктами, а не вопреки им — люди любят выигрывать. Люди любят учиться. Люди любят взаимодействовать с другими людьми. Люди любят быть частью команды. Люди любят управлять. Люди любят, когда им доверяют. Люди любят хорошую работу. Люди любят, когда разрабатываемая ими программа отлично работает.

Пол Чисолм (Paul Chisolm) пишет.

> Я был на **совещании,** на котором один так называемый менеджер контроля качества предложил добавить шесть дополнительных полей в состав HTML-формы, и без того заполненой **данными,** которые к тому же редко когда использовались. Этот самый менеджер **объяснил,** что добавление новых полей необходимо не потому, что данная информация окажется полезной в дальнейшем, а потому, что дополнительные поля позволят СЭКОНОМИТЬ ВРЕМЯ. Моя реакция была следующей: я ударил лбом о твердую поверхность **стола,** за которым проводилось совещание, прямо как мультипликационный герой Warner Brothers, который только что услышал что-то невероятное. После этого я попросил их перестать мне лгать. На сегодняшний день я не знаю, был ли это наименее профессиональный или, наоборот, наиболее профессиональный поступок в моей жизни. Однако мой глазной врач сказал мне, чтобы я больше не стучал головой о стол, так как при этом сетчатка в моем глазу может отсоединиться от глазного дна. (Источник: электронная почта.)

Это достаточно сложно — разработать процесс, в рамках которого краткосрочные личные интересы служат долгосрочным интересам всей команды. Вы можете сколько угодно рассуждать на тему, насколько та или иная методика способствует достижению долгосрочной всеобщей цели, однако как только вы оказываетесь под давлением, вы обнаруживаете, что если методика не способствует решению конкретной проблемы, стоящей перед вами в настоящий момент, вы отбрасываете ее в сторону. Если дисциплина XP не будет удовлетворять краткосрочным личным интересам людей, она обречена на провал.

Некоторым в XP нравится именно то, что эта дисциплина создает у программистов ощущение, будто они занимаются именно тем, чем они занимались бы, если бы их никто не заставлял специально заниматься той или иной проблемой. При этом XP обеспечивает общее развитие проекта в заданном направлении. Мне запомнилась одна образная цитата: «XP соответствует поведению программистов в условиях дикой природы».

Принимаемая ***ответственность*** — ничто не может так повредить работе отдельного человека или всей команды, как жесткое указание, чем именно они должны заниматься. Это особенно справедливо, если то, что предлагается сделать, сделать невозможно. Люди работают эффективнее, только если они чувствуют, что занимались бы этой работой, даже если бы никто их к этому не принуждал. Если человеку приказали выполнять некоторую не устраивающую его работу, в ходе своей деятельности он сможет найти множество способов объяснить свое нежелание и несогласие с этим. Все это пагубно повлияет как на его собственную деятельность, так и на деятельность всей команды.

Альтернативой этого является ответственность, которая не вешается на человека в приказном порядке, а принимается им добровольно. Это не

означает, что вы всегда занимаетесь только тем, чем вам нравится заниматься. Вы являетесь частью команды, и если команда придет к выводу, что некоторая задача требует решения, должен найтись человек, который возьмется за это вне зависимости от того, насколько отталкивающей для него будет эта задача.

Локальная адаптация — вы должны адаптировать то, о чем вы читаете в данной книге, к своим локальным условиям. Это является применением принципа принимаемой ответственности к используемому вами процессу разработки. Адаптация XP не означает, что я должен сказать вам, как вы должны разрабатывать программный продукт. Это означает, что вы должны самостоятельно определить, как вы должны разрабатывать программный продукт. Я могу рассказать вам лишь о том, какие методики я проверил на собственном опыте и при этом убедился, что они неплохо срабатывают. Я могу сообщить вам о возможных последствиях в случае, если вы отклонитесь от использования предлагаемых мною методик. В конце концов, это ваш собственный процесс разработки. Сегодня вы должны определить, как он будет происходить. Вы должны убедиться в том, что принятые решения будут исполняться и завтра. Вы должны модифицировать этот процесс и адаптировать его. Вы не должны думать так: «Теперь я знаю, как следует разрабатывать программы». Вместо этого вы должны сказать себе: «Я должен обдумать все это, а затем уже приступать к программированию». Да, вы должны, но это того стоит.

Путешествие налегке — вы не можете нести с собой гору багажа и при этом двигаться быстро. Вы должны использовать в отношении своего багажа следующие наречия:

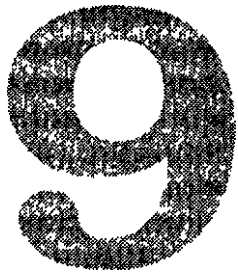
- немного;
- просто;
- ценно.

Члены команды XP должны стать интеллектуальными кочевниками, в любую минуту готовыми быстро упаковать свои пожитки и следовать за пастухом. Пастухом в данном случае является дизайн, который развивается в ином направлении, чем это предполагалось ранее, или заказчик, который хочет развивать проект в ином направлении, чем это предполагалось ранее, или член команды, который решил уйти из проекта, или технология, которая подчас эволюционирует с головокружительной быстротой, или изменяющийся климат, в котором функционирует бизнес.

Как все кочевники, члены команды XP должны привыкнуть путешествовать налегке. Они не должны брать в свое путешествие слишком много — в багаже не должно оставаться ничего, за исключением того, что необходимо для удовлетворения нужд заказчика, то есть тестов и кода.

Откровенные оценки — наше желание контролировать процесс разработки программного обеспечения ведет нас к необходимости оценки. Эта оценка должна быть достаточно точной. Это хорошо, однако похоже на то, что мы вынуждены оценивать на уровне детализации, который не поддерживается имеющимися у нас инструментами. Если у вас нет надежного способа измерения на таком уровне детализации, то лучше сказать: «Это займет две недели, чуть больше или чуть меньше», чем сказать: «14,176 суток». Мы также нуждаемся в единицах измерения, которые соответствуют избранной нами методике работы. Например, количество строк кода становится бессмысленной системой измерения в случае, если код начинает стремительно раздуваться за счет того, что мы обнаружили более эффективные способы программирования.

Обратно к истокам



Мы хотим сделать все, что от нас зависит, для того чтобы получить стабильный, предсказуемый процесс разработки программного продукта. Однако у нас нет времени на что-либо лишнее. Четыре основных рода деятельности, которые составляют собой процесс разработки, — это кодирование, тестирование, слушание и проектирование.

Рассказ об управлении автомобилем. Четыре ценности — коммуникация, простота, обратная связь и храбрость. Двойной набор принципов. Теперь мы готовы приступить к формированию дисциплины разработки программного обеспечения. Вначале надо определить круг решаемых нами проблем. К какой области будут относиться наши предписания? Решением проблем какого сорта мы будем заниматься? Проблемы какого сорта будут игнорироваться нами?

Я вспоминаю, как я впервые научился программировать на BASIC. У меня была пара книг, в которых описывались основы программирования. Я достаточно быстро прочитал их и, когда я закончил, решил приступить к решению проблемы, которая была несколько сложнее, чем примитивные примеры, рассматривавшиеся в этих книгах. Я решил написать игру Star Trek (Звездный путь), похожую на ту, в которую я играл в Lawrence Hall of Science в университете Berkeley, только моя версия должна была бы стать круче.

Чтобы решать упражнения из двух изученных мною книг, я использовал следующий процесс разработки программ: в течение нескольких минут я пристально изучал условие задачи, затем писал код для ее решения, а затем решал возникшие в результате выполнения этого кода проблемы. И вот я уселся за компьютер с твердым намерением написать игру. В течение нескольких минут размышлений у меня не родилось ни одной мысли. Я понятия не имел, как написать приложение объемом более чем

20 строк. Я пересел за письменный стол, решив вначале написать всю программу на бумаге и только после этого перенести ее в компьютер. Я набросал три строки, которые я уже набивал на клавиатуре ранее, а затем опять остановился в мучительном размышлении.

Я чувствовал, что надо сделать что-то еще помимо программирования, но я не знал, что именно.

Что если теперь мы вернемся в то же самое состояние, однако на этот раз на волне обретенного нами опыта? Что мы должны были бы сделать в подобной ситуации? Мы знаем, что нельзя просто «кодировать до тех пор, пока все не будет сделано». Какой деятельностью мы должны будем заняться? Что мы должны получить от каждой из этих деятельностей, если мы осваиваем их заново?

Кодирование

В конце дня у нас должна быть готовая программа. Таким образом, я прихожу к выводу, что кодирование — это как раз та самая деятельность, без которой нам не обойтись. Рисуете ли вы диаграммы, которые автоматически генерируют код, или вы набираете строки в текстовом редакторе, следует считать, что вы кодируете.

Что мы можем извлечь из кода? Наиболее важной вещью является обучение. Обучение происходит следующим образом: у меня появляется мысль, я тестирую ее, чтобы проверить, насколько она хороша. Код — это наиболее удобная вещь для того, чтобы реализовать этот метод на практике. Код не подвластен риторической силе и логике. На код нельзя воздействовать ученой степенью, общественным признанием и высоким окладом. Код просто сидит в вашем компьютере и делает то, что вы ему сказали делать. Если он делает не то, что вы ему сказали делать, — это ваша личная проблема.

Если вы что-то закодировали, у вас появляется возможность понять, какая структура кода будет наилучшей. Внутри кода существуют некоторые признаки, которые сообщают вам о том, что вы пока еще не поняли, какой должна быть необходимая структура.

Код позволяет вам также обмениваться информацией четко, сжато и выразительно. Если у вас есть идея и вы пытаетесь объяснить мне ее, я вполне могу не понять вас. Однако если мы вместе реализуем вашу идею в коде, я могу увидеть в логике написанного вами кода точную формулировку вашей идеи. Опять же, я вижу формулировку идеи не так, как вы видите ее в своей голове, а так, как она выражается для всего остального мира.

Коммуникация подобного рода очень просто преобразуется в обучение. Я вижу вашу идею, и у меня появляется моя собственная, однако мне

сложно объяснить ее на словах, поэтому я, так же как и вы, обращаюсь к кодированию. Так как наши идеи связаны между собой, мы используем связанный код. Вы видите мою идею, и у вас появляется еще одна.

Наконец, код является артефактом, без которого разработка программного продукта абсолютно невозможна. Я слышал истории о системах, в которых исходный код был утерян, и при этом они продолжали функционировать в производственных условиях. Подобные случаи чрезвычайно редки. Чтобы система продолжала жить, для нее должен существовать исходный код.

Раз мы обязаны обладать исходным кодом, значит, мы можем использовать его для максимально возможного количества задач, связанных с разработкой программ. Например, код можно использовать для общения. То есть при помощи кода вы можете объяснить свое тактическое намерение, описать алгоритм, указать на точки возможного будущего расширения или сокращения. Код можно использовать также для того, чтобы объяснить тесты. Тесты используются как для объективного тестирования некоторой операции системы, так и в качестве ценной спецификации системы на всех уровнях.

Тестирование

Английские философы-позитивисты Лок (Locke), Беркли (Berkeley) и Хьюм (Hume) утверждают, что все, чего нельзя измерить, на самом деле не существует. Если речь заходит о коде, я с ними полностью согласен. Возможности программного продукта, которые нельзя продемонстрировать с использованием тестов, просто не существуют. Я запросто могу обмануть самого себя, убедив себя в том, что то, что я написал, есть то, что я имел в виду. Я также вполне могу обмануть себя в том, что то, что я имел в виду, является тем, что я должен был иметь в виду. Поэтому я не должен верить ничему, что я написал до тех пор, пока я не напишу для этого тесты. Тесты позволяют мне думать о том, что я хочу, вне зависимости от того, как это реализовано. Если я что-либо реализовал, тесты сообщают мне о моем представлении о том, что я реализовал.

Многие люди думают об автоматических тестах в контексте тестирования функциональности — то есть вычисления чисел. Чем более опытным я становлюсь в деле написания тестов, тем яснее для меня становится, что я могу разрабатывать тесты для тестирования нефункциональных требований, например для тестирования производительности, или соответствия некоторым стандартам кодирования.

Эрих Гамма (Erich Gamma) придумал термин «инфицированный тестами» (Test Infected) для описания людей, которые не приступают к ко-

дированию до тех пор, пока у них не будет набор тестов для проверки разрабатываемого кода. Тесты сообщают вам о том, что ваша работа завершена, — когда все тесты сработали, считайте, что на данный момент кодирование успешно завершено. Когда вы больше не можете придумать ни одного теста, можете считать, что вы завершили работу.

Тесты — это ресурс и ответственность. Вы не можете написать всего один тест, добиться его работы и объявить, что на этом ваша работа закончена. Вы несете ответственность за разработку всех тестов, которые могут не сработать, которые вы только можете себе представить. Через некоторое время вы получите неплохое предощущение относительно тестов — если эти два теста сработали, значит, можно со всей уверенностью заключить, что этот третий тест также сработает, и его вовсе не обязательно писать. Конечно же, именно такие рассуждения ведут к появлению ошибок в программах, поэтому вы должны быть очень осторожными в этом отношении. Если в дальнейшем возникают проблемы, которые можно было бы обнаружить раньше, если бы только вы написали вовремя этот третий тест, вам необходимо должным образом воспринять этот горький опыт и в следующий раз заставить себя не отказываться от разработки подобного третьего теста.

Большая часть программного обеспечения разрабатывается без использования автоматического тестирования. Очевидно, что автоматические тесты — это необязательная составная часть разработки. Почему же я все-таки включил тестирование в список важнейших родов деятельности при разработке программного продукта? На этот вопрос я готов дать два ответа: один для краткосрочной перспективы, другой — для долгосрочной.

В долгосрочной перспективе тесты позволяют программе жить дольше (если конечно они работают и должным образом поддерживаются в рабочем состоянии). Если у вас есть тесты, вы можете вносить в программу более значительные изменения в течение более длительного времени. Если у вас нет тестов, вы теряете такую возможность, так как любое изменение перестает быть предсказуемым и может обернуться катастрофой. Если вы продолжаете писать тесты, со временем ваша уверенность в системе увеличивается.

Один из принципов предписывает, что необходимо работать совместно с природой человека, а не против нее. Если тестирование подкрепляется только лишь одним долгосрочным аргументом, об этом аргументе можно легко позабыть. В этом случае многие будут заниматься тестированием только потому, что они обязаны это делать, или потому, что кто-то тщательно контролирует правильность их работы. Как только внимание надсмотрщика ослабевает, или в случае, если сроки сдачи работы

стремительно приближаются, разработка новых тестов прекращается, уже имеющиеся тесты перестают запускаться, и в результате вся система разваливается на части. Таким образом, если мы хотим работать сообразно с человеческой природой и при этом мы хотим обеспечить тестирование, мы должны найти для тестирования краткосрочную эгоистическую причину.

К счастью, такая краткосрочная причина существует. Программирование в случае, если вы используете тесты, — это более приятный процесс, чем программирование без тестов. Вы кодируете со значительно большей уверенностью. У вас никогда не возникает страха наподобие: «В это место системы надо бы внести изменения, но вдруг я что-нибудь сломаю?» Вы просто меняете код, щелкаете на кнопке, запускаются все тесты, если при этом на экране появляется зеленый цвет, вы можете продолжать работу с еще большей уверенностью.

Я помню, как я занимался этим на публичной программистской демонстрации. Каждый раз, когда я отворачивался от аудитории, чтобы продолжить программирование, я машинально щелкал на кнопке тестирования. Я не менял никакого кода. Абсолютно все в рабочей среде оставалось неизменным. Зачем же я раз за разом щелкал на тестирующей кнопке? Я всего лишь хотел получить заряд уверенности. Когда я в очередной раз видел, что все тесты по-прежнему срабатывают и ничего в системе не нарушено, я получал такой заряд!

Совместное программирование и тестирование выполняется быстрее, чем просто программирование. Когда я только начинал использовать данную методику, я не ожидал такого эффекта, однако я со всей очевидностью заметил его. Мало того, помимо меня об этом сообщает множество других людей. Возможно, отказавшись от тестирования, вы сможете сэкономить полчаса, однако как только вы привыкнете к использованию тестов, вы быстро отметите разницу в производительности. Выигрыш в производительности получается за счет того, что уменьшается время, необходимое вам для отладки, — вместо того, чтобы заниматься поиском ошибки в течение часа, вы обнаруживаете ее в течение нескольких минут. Иногда вы никак не можете добиться, чтобы тест срабатывал. Это означает, что, скорее всего, вы столкнулись с существенно более крупной проблемой. В этом случае вы должны сделать шаг назад и убедиться в том, что все ваши тесты корректны. Вы также должны проверить весь дизайн системы — возможно, он требует серьезного пересмотра.

Однако существует опасность. Плохо организованное тестирование — это все равно, что розовые очки, сквозь которые вы смотрите на свою

систему. Вы получаете ложную уверенность в том, что ваша система в порядке, — еще бы, ведь все тесты срабатывают. Вы продолжаете движение вперед, не подозревая, что оставляете позади себя ловушку. Как только вы в следующий раз пойдете этим же путем, ловушка может сработать.

Весь трюк, связанный с тестированием, заключается в нахождении приемлемого для вас уровня дефектов. Если вы в состоянии позволить себе одну жалобу со стороны заказчика в течение месяца, вкладывайте ресурсы в улучшение тестирования и улучшайте его до тех пор, пока не достигнете желаемого уровня. Затем, используя полученный стандарт тестирования, продолжайте движение вперед, так как состояние системы считается отличным в случае, если все тесты срабатывают.

Заглядывая вперед, отмечу, что в дальнейшем разговор пойдет о двух наборах тестов. Тесты модулей (unit tests) разрабатываются программистами для того, чтобы убедиться в корректной работе разрабатываемого ими кода. Функциональные тесты (functional tests) разрабатываются (или по крайней мере специфицируются) заказчиками для того, чтобы убедиться в том, что система как единое целое работает именно так, как она должна работать.

Таким образом, для тестов существуют две аудиенции. Программисты должны оформить свою уверенность в разрабатываемом коде в реальную форму для того, чтобы кто-то другой также мог получить эту уверенность. Заказчики должны подготовить набор тестов для того, чтобы получить от системы подтверждение своей уверенности: «Замечательно, я полагаю, что если вы сможете добиться срабатывания всех этих тестов, это значит, что система работает».

Слушание

Программисты ничего не знают. Говоря точнее, программисты не знают ничего такого, что является интересным для бизнесменов. Если бы бизнесмены могли обойтись без программистов, они бы вышвырнули нас вон в одну секунду.

К чему я веду? Если вы решили тестировать, вы должны получить откуда-либо ожидаемые ответы. Так как вы (программист) ничего не знаете, вы должны спросить у кого-то еще. Они сообщат вам, какие ответы являются ожидаемыми и какие *случаи* являются необычными с точки зрения бизнеса.

Если вы намерены задать вопрос, вы должны быть готовыми услышать ответ. Таким образом, слушание — это третий род деятельности в рамках разработки программного обеспечения.

Программисты должны слушать с большим вниманием. Они должны услышать от заказчика суть бизнес-проблемы. Они должны помочь заказчику понять, что является простым, а что — сложным, это можно считать активной формой слушания. Обратная связь от программистов к заказчику помогает заказчику самому лучше понять суть стоящей перед ним проблемы.

Если вы просто скажете участникам проекта: «Вы должны слушать друг друга, и вы должны слушать заказчика», то этим вы не добьетесь желаемого результата. Многие уже попробовали это и пришли к выводу, что подобные простые директивы не срабатывают. Мы должны найти способ структурировать коммуникацию так, чтобы в результате обсуждения речь шла именно о тех вещах, которые нуждаются в обсуждении, это должно происходить именно в то время, когда возникает надобность в подобном обсуждении, и именно в том объеме, в котором эти вещи должны обсуждаться. Кроме того, разрабатываемые нами правила должны избавлять команду от коммуникации, которая только мешает дальнейшему развитию проекта.

Проектирование

Почему нельзя просто слушать, затем писать тестовый случай, затем заставить его работать, затем опять слушать, опять писать тестовый случай и опять заставить его работать, и так далее? Потому что мы знаем, что это не сработает. Конечно, вы можете попробовать действовать именно так в течение некоторого времени. Вы можете даже действовать так в течение достаточно длительного времени. Однако в определенный момент вы не сможете продолжать работать над проектом. Вы попадете в ситуацию, когда для того, чтобы заставить работать некоторый тестовый случай, вам придется нарушить работу другого тестового случая. Или для того, чтобы заставить работать тестовый случай, вам придется затратить столько усилий, что это перестанет быть экономически выгодным. Энтропия проглатывает еще одну жертву.

Единственным способом избежать этого является проектирование. Проектирование — это создание структуры, которая организует логику в системе. Хороший дизайн организует логику так, что для внесения изменений в одну часть системы вам не нужно обязательно вносить изменения в другую часть системы. Хороший дизайн предусматривает, что каждый логический раздел системы оформляется в виде самостоятельного независимого фрагмента программы. Хороший дизайн размещает логику рядом с данными, в отношении которых она действует. Хороший дизайн позволяет расширять систему, модифицируя только лишь одно ее место.

Плохой дизайн — это прямая противоположность. Если в рамках плохого дизайна вы пытаетесь внести в систему концептуальное изменение, вам приходится вносить изменения сразу в несколько разных мест системы. В рамках плохого дизайна одна и та же логика дублируется в нескольких местах. Со временем затраты, вызванные плохим дизайном, становятся чрезмерно большими. Вы просто забываете о том, в какие места системы необходимо внести все связанные между собой изменения. Вы не можете добавить в систему новую функцию, не нарушив при этом работу одной из уже существующих функций.

Сложность — это еще одна причина плохого дизайна. Если для того, чтобы выяснить, что же все-таки происходит, дизайн предусматривает четыре уровня перенаправления и если эти уровни не обладают функциональным или смысловым предназначением, значит, дизайн плохой.

Таким образом, последним родом деятельности, который мы должны структурировать в рамках разрабатываемой нами новой дисциплины, — это проектирование, или, по-другому, формирование дизайна. Мы должны сформировать контекст, в рамках которого создается только хороший дизайн, а плохой дизайн исправляется. Кроме того, в рамках этого контекста о текущем дизайне системы знает каждый, кому это необходимо.

Как будет показано в последующих главах, методы формирования дизайна в XP существенно отличаются от методов, традиционно используемых в рамках многих других дисциплин разработки программного обеспечения. В рамках XP проектирование является частью ежедневной работы каждого из программистов. Программисты XP занимаются проектированием прямо в процессе кодирования. Однако вне зависимости от стратегии, которая используется для получения хорошего дизайна, в процессе разработки программного продукта проектирование не выполняется по желанию. Это неотъемлемая часть любого программного проекта, и для того, чтобы сделать разработку программы эффективной, вы должны уделить проектированию очень серьезное внимание.

Заключение

Итак, вы кодируете потому, что без этого вы вообще не сможете получить какого-либо результата. Вы тестируете потому, что без этого вы не сможете определить, завершена ли ваша работа. Вы слушаете потому, что без этого вы не сможете понять, что собственно кодировать и что тестировать. Наконец, вы проектируете для того, чтобы получить возможность продолжать кодировать, тестировать и слушать неограниченное время. Вот

и все. Именно эти четыре рода деятельности нам и предстоит структурировать в рамках нашей новой дисциплины:

- кодирование;
- тестирование;
- слушание;
- проектирование.

Часть 2

Решение

Можно считать, что мы подготовили сцену, на которой должна появиться разрабатываемая нами дисциплина. Мы познакомились с проблемой, которую нам предстоит решить. Мы поняли, что для формирования решения этой проблемы нам необходимо определить, каким образом должны осуществляться кодирование, тестирование, слушание и проектирование — четыре базовых рода деятельности, которыми приходится заниматься в рамках любого программного проекта. Мы обладаем набором направляющих ценностей и принципов, которые помогут нам в выборе стратегий для каждого из этих родов деятельности. Наконец, мы знаем, что кривая затрат должна быть пологой, а не экспоненциальной, благодаря этому мы сможем организовать эффективную работу над программным проектом в рамках XP.

Краткий обзор

Мы будем опираться на симбиоз взаимодействующих между собой методик. Методик, некоторые из которых были забыты десятилетия назад как непрактичные и наивные.

Вот исходные материалы, из которых нам предстоит построить новую дисциплину разработки программного обеспечения:

- история об управлении автомобилем;
- четыре ценности — коммуникация, простота, обратная связь и храбрость;
- принципы;
- четыре базовые активности — кодирование, тестирование, слушание и проектирование.

Наша задача — структурировать четыре активности. Мы должны не только структурировать активности, мы должны сделать это в соответствии с длинным списком подчас противоречивых принципов. В то же время мы должны попытаться улучшить экономическую производительность разработки программного обеспечения таким образом, чтобы все получили возможность слушать.

Нет проблем.

Э-э-э...

Целью данной книги является объяснение того, как работают входящие в XP методики, поэтому в данной главе я бегло перечислю основные группы используемых в рамках XP методик. В следующей главе я покажу, как подобные смехотворно простые решения могут дать столь значительный результат. Там, где некоторая методика слаба, сила остальных методик покрывает недостатки слабой. В последующих главах некоторые темы будут рассмотрены более детально.

Для начала перечислю все методики.

- *Игра в планирование* (planning game) — быстро определяет перечень задач (объем работ), которые необходимо реализовать в следующей версии продукта. Для этого рассматриваются бизнес-приоритеты и технические оценки. Если со временем план перестает соответствовать действительности, происходит обновление плана.
- *Небольшие версии* (small releases) — самая первая упрощенная версия системы быстро вводится в эксплуатацию, после этого через относительно короткие промежутки времени происходит выпуск версии за версией.
- *Метафора* (metaphor) — эта простая общедоступная и общеизвестная история, которая коротко описывает, как работает вся система. Эта история управляет всем процессом разработки.
- *Простой дизайн* (simple design) — в каждый момент времени система должна быть спроектирована так просто, как это возможно. Чрезмерная сложность устраняется, как только ее обнаруживают.
- *Тестирование* (testing) — программисты постоянно пишут тесты для модулей. Для того чтобы разработка продолжалась, все тесты должны срабатывать. Заказчики пишут тесты, которые демонстрируют работоспособность и завершенность той или иной возможности системы.
- *Переработка* (refactoring) — программисты реструктурируют систему, не изменяя при этом ее поведения. При этом они устраняют дублирование кода, улучшают коммуникацию, упрощают код и повышают его гибкость.
- *Программирование парами* (pair programming) — весь разрабатываемый код пишется двумя программистами на одном компьютере.
- *Коллективное владение* (collective ownership) — в любой момент времени любой член команды может изменить любой код в любом месте системы.
- *Непрерывная интеграция* (continuous integration) — система интегрируется и собирается множество раз в день. Это происходит каждый раз, когда завершается решение очередной задачи.
- *40-часовая неделя* (40-hour week) — программисты работают не более 40 часов в неделю. Это правило. Никогда нельзя работать сверхурочно две недели подряд.
- *Заказчик на месте разработки* (on-site customer) — в состав команды входит реальный живой пользователь системы. Он доступен в течение всего рабочего дня и способен отвечать на вопросы о системе.

- *Стандарты кодирования* (coding standards) — программисты пишут весь код в соответствии с правилами, которые обеспечивают коммуникацию при помощи кода.

В данной главе кратко рассказывается о том, что подразумевается под каждой из этих методик. В следующей главе («Как это работает?») мы рассмотрим взаимосвязи между методиками, благодаря которым слабость одной методики нейтрализуется силой другой методики.

Игра в планирование

Ни соображения бизнеса, ни технические соотношения не должны превалировать. Разработка программного обеспечения — это всегда эволюционирующий диалог между желаемым и возможным. Природа этого диалога состоит в том, что в его процессе изменяется как то, что кажется возможным, так и то, что кажется желаемым.

Представители бизнеса должны принимать решения в следующих областях.

- *Объем работ* — какую часть проблемы достаточно решить для того, чтобы систему можно было эксплуатировать в реальных производственных условиях? Представитель бизнеса должен быть способен определить, какого объема будет недостаточно и какой объем будет чрезмерным.
- *Приоритет* — если с самого начала вы можете реализовать только возможности A или B, то какую из них следует реализовать в первую очередь? Ответ на этот вопрос должен быть определен в первую очередь представителем бизнеса, а не программистом.
- *Композиция версий* — как много или как мало должно быть сделано для того, чтобы бизнес лучше шел с программным обеспечением, чем без него? Программистская интуиция зачастую ошибается в отношении данного вопроса.
- *Сроки выпуска версий* — в какие важные даты очередные версии программного продукта должны появляться в производстве?

Бизнес не может принимать решения в вакууме. Разработчики должны сформировать набор технических решений, которые должны стать исходным материалом при формировании бизнес-решений.

Разработчики должны принимать решения в следующих областях.

- *Оценка* — как много времени потребуется для того, чтобы реализовать ту или иную возможность?
- *Последствия* — существует набор бизнес-решений, которые следует формировать, только ознакомившись с технологическими по-

следствиями. Хорошим примером является выбор той или иной технологии управления базой данных. Бизнесу лучше иметь дело с крупной компанией, а не с новичками, однако и здесь существует набор факторов, которые необходимо тщательно изучить, так как, возможно, сотрудничество с компанией, появившейся на рынке недавно, оправданно по тем или иным причинам. Возможно, свежая, недавно разработанная система управления базой данных дает двукратный рост производительности. Возможно, разработка решения на основе этой базы данных обойдется бизнесу в два раза дешевле. Таким образом, риск, связанный с использованием новой технологии управления базой данных, вполне оправдан. А возможно, и нет. Разработчики должны объяснить последствия того или иного решения.

- *Процесс* — как будет организована команда разработчиков и как будет организована работа этой команды? Команда должна соответствовать культуре, в рамках которой будет осуществляться разработка, однако при этом не следует забывать, что ваша основная задача — получить качественный программный продукт, а не следить за чистотой культуры разработки.
- *Подробный график работ* — какие истории заказчика должны быть реализованы в первую очередь в рамках работы над очередной версией продукта? Программисты должны обладать свободой при решении вопроса о том, какие самые рискованные сегменты разработки должны быть реализованы в первую очередь. Благодаря этому снижается общий риск разработки. В рамках этого ограничения по-прежнему следует в первую очередь работать над наиболее приоритетными для бизнеса задачами. Благодаря этому снижается вероятность того, что реализация чрезвычайно важных для бизнеса возможностей системы будет отложена до следующей версии.

Небольшие версии

Каждая версия должна быть настолько маленькой, насколько это возможно. В ней должны быть реализованы наиболее ценные для бизнеса требования. В целом версия должна быть логически завершенной и работоспособной, то есть вы не можете реализовать некоторую возможность только наполовину и внедрить ее в производство только для того, чтобы сократить время работы над версией.

Лучше планировать на месяц или два вперед, чем планировать на полгода или год вперед. Компания, которая за один раз передает в руки заказчика достаточно крупный программный продукт, не может выпускать очередные версии этого продукта достаточно часто. Эта компания долж-

на сократить время работы над очередной версией настолько, насколько это возможно.

Метафора

Каждый программный проект XP направляется при помощи единой всеобъемлющей метафоры. Иногда эта метафора выглядит «наивной», как, например, система управления контрактами, о которой рассказывается в терминах контрактов, заказчиков и индоссаментов. Иногда метафора требует дополнительных разъяснений, например, требуется отметить, что компьютер должен рассматриваться как рабочий стол или вычисление пенсии должно выглядеть как электронная таблица. Все это метафоры, так как на самом деле мы не говорим буквально, что «система — это электронная таблица». Метафора просто помогает каждому участнику проекта понять базовые элементы программы и то, как они взаимосвязаны.

Слова, которые используются для идентификации технических элементов системы, должны постоянно заимствоваться из выбранной метафоры. По мере того как идет работа над проектом, метафора развивается и вся команда продолжает ее анализировать, получая при этом новые источники вдохновения.

В XP метафора во многом заменяет собой то, что другие люди называют термином «архитектура». Проблема состоит в том, что архитектура — это, как правило, огромная по размерам схема системы, которая не дает представления о ее целостности. Архитектура — это большие прямоугольники и соединения между ними.

Конечно же, вы можете сказать: «плохо сформированная архитектура — это плохо». Однако нам требуется подчеркнуть саму цель, для которой формируется архитектура, а это значит, что мы должны предоставить каждому участнику проекта связную историю о строении и функционировании системы. Эта история должна быть изложена на языке, понятном как технарям, так и бизнесменам. В рамках этой истории будет осуществляться работа над проектом. Спросив о метафоре, мы получаем в ответ сведения об архитектуре, причем эти сведения передаются нам в такой форме, которая удобна для общения и обдумывания.

Простой дизайн

В каждый момент времени правильным является дизайн системы, в рамках которого:

1. Выполняются все тесты.
2. Нет дублирующейся логики. (Опасайтесь скрытого дублирования, например, применения параллельных иерархий классов.)

3. Выражается каждая из идей, важных для программистов.
4. Существует наименьшее возможное количество классов и методов.

Каждый фрагмент дизайна системы должен доказать свое право на существование на основании всех перечисленных правил. Эдвард Туфт (Edward Tufte) придумал упражнение для разработчиков графов — нарисуйте граф так, как хотите, затем стирайте до тех пор, пока вы не удаляете из графа полезную информацию. Если вы не можете больше продолжать стирание, значит, перед вами наиболее удачный дизайн для графа. Простой дизайн программной системы можно сформировать точно таким же образом — уберите из системы все элементы, какие вы сможете убрать, не нарушив при этом правил 1-3.

Edward Tufte, *The Visual Display of Quantitative Information* (Визуальное отображение численной информации), Graphics Press, 1992.

Этот совет противоположен тому, что обычно приходится слышать программистам: «Реализуйте для сегодняшнего дня, а проектируйте — для завтрашнего». Однако если вы уверены, что будущее — в неопределенности, если вы верите в то, что завтра вы можете сменить направление своих мыслей и не платить за это слишком дорого, значит, включение в дизайн функциональности только на основании абстрактных размышлений — это безумство. Добавляйте в дизайн то, что вам нужно, только тогда, когда это действительно вам нужно.

Тестирование

Любая возможность программы, для которой нет автоматических тестов, просто не существует. Программисты пишут тесты модулей, благодаря чему их уверенность в правильности функционирования программы становится частью самой программы. Заказчики пишут функциональные тесты, благодаря чему их уверенность в функционировании программы также становится частью программы. В результате всеобщая уверенность в работоспособности программы со временем все возрастает и возрастает. Эта уверенность выражена в наборе тестов, количество которых увеличивается и которые продолжают функционировать по мере продолжения работы над программой. Благодаря этому со временем программа становится не менее, а более приспособленной для внесения в нее изменений.

Нет необходимости писать тесты для каждого разрабатываемого вами метода, проверять надо только производственные методы, которые могут не сработать. Иногда вы тратите усилия только на то, чтобы понять, возможно ли в процессе функционирования кода возникновение той или иной ситуации. В течение получаса вы анализируете код. Да, это возможно. Теперь вы отбрасываете код и начинаете писать его заново — начиная с тестов.

Переработка

Когда программисты приступают к реализации некоторой возможности программы, они всегда задаются вопросом, существует ли способ изменения имеющейся программы для того, чтобы упростить добавление в нее требуемой новой возможности? После того как возможность добавлена, программисты спрашивают себя, можно ли теперь упростить программу и при этом обеспечить выполнение всех тестов? Это и называется переработкой кода (refactoring).

Обратите внимание, это означает, что иногда вы должны сделать больше работы, чем это на самом деле необходимо для того, чтобы добавить в программу новую возможность. Однако, работая в этом ключе, вы обеспечиваете снижение затрат, связанных с добавлением в программу последующих возможностей. Переработав код, вы упрощаете его дальнейшую модернизацию. Вы не выполняете переработку исходя из предварительных нечетких размышлений, напротив, вы перерабатываете код тогда, когда система нуждается в этом. Когда при добавлении новой возможности вы дублируете код, это означает, что система просит вас выполнить переработку.

Если программист видит некоторый «черновой» способ обеспечить выполнение теста, для реализации которого потребуется одна минута, и, помимо этого, он также видит другой, более элегантный способ обеспечить выполнение теста, предусматривающий также упрощение дизайна, но для реализации которого требуется десять минут, программист должен предпочесть второй способ, то есть потратить больше времени, но в результате получить более простой и более элегантный дизайн. К счастью, в рамках XP вы получаете возможность вносить в дизайн системы любые, даже самые радикальные изменения, делая это небольшими, малорискованными шагами.

Программирование парами

Весь код системы вплоть до ее внедрения в производство пишется парами программистов, каждая из которых работает на одном компьютере, оснащенном одной клавиатурой и одной мышью.

В каждой паре существуют две роли. Один партнер, в руках которого находится клавиатура и мышь, думает о том, как прямо сейчас реализовать некоторый метод самым лучшим образом. Второй партнер думает более стратегически:

- Сработает ли используемый подход в целом?
- Какими могут быть другие, еще не рассмотренные тестовые случаи?

- Существуют ли какие-либо способы упростить всю систему таким образом, что текущая проблема просто исчезнет?

Состав пар меняется динамически. Партнеры, которые утром входили в состав одной пары, днем могут стать членами совершенно других пар. Если вы отвечаете за решение задачи в области, которая является для вас малознакомой, вы можете попросить составить вам компанию кого-либо, кто недавно работал в этой области. Скорее всего, вы побываете в паре с каждым из членов вашей команды.

Коллективное владение

Любой член команды, который видит возможность добавить что-либо в любой раздел кода системы, может сделать это в любой подходящий для этого момент времени.

Сравните это с двумя другими моделями владения кода — полное отсутствие владения и индивидуальное владение. В давние времена различными кусками кода программы никто не владел. Если кто-либо желал изменить какой-либо код, он мог сделать это в соответствии со своими собственными пожеланиями. Результатом был хаос, в особенности если приходилось иметь дело с объектами, в которых взаимосвязь между строкой кода в одном месте и строкой кода в другом месте нельзя было в точности установить статически. Код разрастался очень быстро, и с такой же скоростью он стремительно терял стабильность.

Чтобы подвести ситуацию под контроль, программисты стали использовать индивидуальное владение кодом. Единственным человеком, кто обладал правом внесения в некоторый фрагмент кода изменений, являлся официальный владелец этого кода. Если кто-либо, не являющийся владельцем, видел, что код необходимо изменить, он должен был обратиться с соответствующей просьбой к владельцу. В результате такой практики действительный код системы начинал расходиться с тем, каким его хотели бы видеть работающие в рамках проекта программисты. Изменение кода в рамках подобного подхода превращалось в своего рода бюрократическую процедуру — люди начинали избегать обращаться к владельцу кода для того, чтобы внести в код желаемые изменения, вместо этого они предпочитали работать с тем, что есть. В конце концов, внести изменение требуется прямо сейчас, а не спустя некоторое время. Таким образом, код оставался относительно стабильным, однако он не эволюционировал с достаточно большой скоростью. А когда владелец кода находил другую работу и уходил из команды... возникали серьезные проблемы.

В рамках XP ответственность за весь код системы лежит на всех членах команды. Нельзя сказать, что каждый член команды хорошо знает

каждую часть кода, однако можно сказать, что каждый член команды знает, по крайней мере, что-то о каждой части. Если пара программистов работает над решением некоторой задачи и видит, что для упрощения работы требуется внести модификации в некоторую часть кода, тем самым улучшив этот код, изменения вносятся немедленно, благодаря чему решаемая этой парой задача упрощается.

Постоянно продолжающаяся интеграция

Код интегрируется и тестируется каждые несколько часов, минимум один раз в день. Для того чтобы обеспечить это, проще всего выделить для этой цели один специально предназначенный для этого компьютер. Когда этот компьютер освобождается, пара, у которой имеется код, подлежащий интеграции, садится за интеграционный компьютер, загружает текущую версию системы, добавляет в нее свои собственные изменения (проверяя и устраняя любые несоответствия и конфликты) и запускает тесты до тех пор, пока все они не сработают (все 100% тестов).

Интеграция одного набора изменений за один раз отлично срабатывает, так как становится очевидным, кто именно должен исправить тест, который не сработал, — мы должны, так как, должно быть, именно мы его сломали. Это связано с тем, что предыдущая пара, которая выполняла интеграцию, добилась срабатывания всех 100% тестов. И если мы не добьемся срабатывания всех 100% тестов, мы должны выкинуть из системы все, что мы написали, и начать решать задачу заново, так как очевидно, что в этом случае, приступая к решению, мы просто не знали всего того, что требуется для разработки требуемого кода (возможно, мы не знаем всего необходимого и сейчас).

40-часовая рабочая неделя

Я хочу быть свежим и энергичным каждое утро, равно как и уставшим и удовлетворенным каждый вечер. Каждую пятницу я хочу быть уставшим и удовлетворенным настолько, чтобы в течение последующих двух дней чувствовать себя в состоянии думать о чем-либо, не связанном с работой. После этого, в понедельник, я хочу приходить на работу с горящими глазами и головой, наполненной идеями.

Конечно же, для этого необязательно, чтобы рабочих часов в неделе было бы ровно 40. Разные люди способны эффективно работать в течение различного времени. Один человек не может концентрировать свое внимание дольше, чем в течение 35 рабочих часов, другой способен успешно действовать в течение 45 часов в неделю. Но никто не способен работать по 60 часов в неделю на протяжении многих недель и при этом

оставаться свежим, творческим, внимательным и уверенным в своих силах. Ни в коем случае не делайте этого!

Работа во внеурочное время — это признак серьезных проблем в проекте. В рамках XP действует очень простое правило — нельзя работать во внеурочное время две недели подряд. В течение одной недели можно поднапрячься и поработать несколько лишних часов. Но если в очередной понедельник вы приходите на работу и объявляете: «Чтобы достичь поставленных целей, мы должны снова работать допоздна», это означает, что у вас возникли проблемы, которые вы не сможете решить простым увеличением рабочего времени.

Отпуск является близкой к этому темой для обсуждения. Европейцы часто отдыхают в течение двух, трех или четырех последовательных недель. Американцы редко когда берут для отдыха больше чем несколько дней подряд. Если бы это была моя компания, я настаивал бы на том, чтобы люди отдыхали в течение двух последовательных недель ежегодно и при этом у них в запасе было бы не менее одной или двух недель дополнительно для более коротких отпусков.

Заказчик на месте разработки

Для ответов на возникающие вопросы, решения споров и установки мелкомасштабных приоритетов рядом с командой разработчиков должен постоянно находиться реальный заказчик. Под термином «реальный заказчик» я подразумеваю человека, который действительно пользуется системой, когда эта система работает в производственных условиях. Например, если вы разрабатываете систему обслуживания клиентов, заказчиком будет служащий по работе с клиентами, пользующийся этой системой. Если вы разрабатываете систему обмена долговыми обязательствами, заказчиком будет биржевой брокер, работающий с этой системой.

Основным препятствием для воплощения этого правила является то обстоятельство, что реальные пользователи разрабатываемой системы обходятся для заказчика слишком дорого в смысле рабочего времени. Иногда менеджерам заказчика жалко жертвовать одним из реальных служащих компании только для того, чтобы он постоянно находился вместе с разработчиками и отвечал на их вопросы. Менеджеры должны решить, что является для них более важным — ускорение и повышение качества разработки или рабочее время одного или двух сотрудников. Если программная система не приносит бизнесу больше, чем приносит ему один из сотрудников предприятия, скорее всего, такая система не стоит того, чтобы ее разрабатывать и внедрять на производстве.

Кроме того, утверждение, что представитель заказчика, работающий вместе с командой, не может заниматься некоторыми своими повседнев-

ными делами, на самом деле не верно. Даже такие творческие люди, как программисты, не могут генерировать вопросы непрерывно в течение 40 часов каждую неделю. Конечно, сотрудник предприятия-заказчика обладает тем недостатком, что он физически удален от тех людей, с которыми он должен взаимодействовать, однако при этом у него будет достаточно времени для выполнения некоторой своей обычной работы.

Недостатком такого подхода является то, что в случае, если проект в конце концов умирает, то сотни часов, которые сотрудник предприятия-заказчика потратил на помощь и консультации разработчиков, оказываются временем, потраченным впустую. Получается, что заказчик потерял работу, которую его сотрудник делал вместе с разработчиками, а также он потерял работу, которую его сотрудник мог бы сделать в соответствии со своими прямыми обязанностями, если бы его рабочим временем не пожертвовали ради поддержки разработки проекта. XP делает все возможное для того, чтобы проект не умирал.

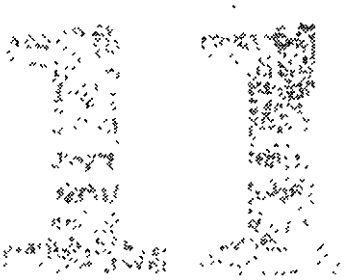
Однажды я работал над одним проектом, и фирма-заказчик с большой неохотой выделила нам одного реального сотрудника, но «только ненадолго». После того как мы завершили первый этап разработки и успешно внедрили первую версию системы на производстве, когда стало очевидным, что система может продолжать эволюционирование, менеджеры заказчика выделили нам трех реальных сотрудников. Компания-заказчик пришла к выводу, что сможет получить от системы больше прибыли, если пожертвует большим количеством своих сотрудников.

Стандарты кодирования

Если мы собираемся перебрасывать программистов с разработки одной части системы на разработку другой части системы, обеспечивать постоянную смену партнеров в парах по несколько раз на дню, обеспечивать постоянную переработку кода программистами, которые этот код не писали, мы просто не можем позволить себе использовать в рамках одного проекта применение нескольких разных стилей кодирования. Спустя некоторое время работы над системой уже нельзя будет сказать точно, какой именно член команды написал тот или иной код.

Стандарт должен требовать от разработчиков приложения минимально возможных усилий для реализации той или иной возможности. Он должен способствовать воплощению на практике правила трех O — Once and Only Once (запрет на дублирование кода). Стандарт должен способствовать коммуникации. Наконец, стандарт должен быть добровольно воспринят всей командой разработчиков.

Как это работает?



Методики поддерживают друг друга. Слабость одной из них покрывается за счет силы других.

Подождите-ка минутку! Ни одна из перечисленных ранее методик не является уникальной или оригинальной. Все они используются уже давно, с самого начала эпохи программирования. От многих из этих методик в свое время отказались в пользу других, более сложных и более высокоуровневых, так как их слабость стала очевидной. Не является ли XP слишком упрощенным взглядом на разработку программ? Прежде чем мы продолжим, мы должны убедиться в том, что эти рассмотренные ранее простые методики не уничтожат нас точно так же, как они уничтожили множество программных проектов десятилетия назад.

Благодаря тому что у нас появилась возможность заменить экспоненциальную кривую роста затрат на пологую, почти асимптотическую линию, мы вновь можем с успехом использовать рассмотренные ранее методики и тем самым существенно повысить эффективность нашей работы. Каждая из этих методик, как и раньше, обладает некоторыми недостатками, но что, если мы попытаемся устранить недостатки одной методики за счет достоинств других методик? Должно быть, мы сможем существенно упростить себе работу.

В данной главе мы вновь рассмотрим описанные ранее методики, но на этот раз с несколько другой стороны. Мы попытаемся проанализировать, что делает ту или иную методику непригодной для использования. Мы также продемонстрируем, каким образом недостатки одной методики компенсируются достоинствами других методик. В данной главе показывается также, каким образом весь механизм XP может работать на благо разработки программного продукта.

Игра в планирование

Вы не можете начать работу над проектом, имея на руках лишь грубый, нечеткий план предстоящей работы. Вы не можете постоянно обновлять ваш план — это потребовало бы слишком много времени, и в результате заказчики остались бы недовольны. Однако в XP:

- заказчики выполняют обновление плана самостоятельно, на основании оценок, которые делаются программистами;
- в самом начале работы вы обладаете планом, достаточно четким для того, чтобы дать заказчикам представление о том, чем может стать для них разрабатываемая система через пару лет;
- вы выпускаете продукт небольшими версиями, благодаря чему любая ошибка в плане будет портить жизнь заказчику лишь в течение нескольких недель или месяцев, но не более того;
- заказчик в лице своего представителя постоянно находится рядом с разработчиками, благодаря чему он может быстро наметить потенциальные изменения в плане и быстро внести в него необходимые улучшения.

При выполнении этих условий вы, скорее всего, можете без опасений приступать к разработке, имея на руках лишь упрощенный план, с намерением в дальнейшем постоянно пересматривать его и вносить в него изменения.

Небольшие версии

Вы не можете приступить к эксплуатации системы уже через несколько месяцев работы над проектом. Вы определенно не можете выпускать новые версии системы каждый день или каждые несколько месяцев. Однако в XP:

- игра в планирование помогает вам работать над наиболее важными историями, поэтому даже небольшая система в состоянии приносить пользу бизнесу;
- вы занимаетесь интеграцией системы постоянно и ежедневно, поэтому затраты, связанные с формированием очередной полноценной рабочей версии, существенно сокращаются;
- в процессе разработки вы постоянно выполняете тестирование системы, благодаря этому количество дефектов в системе настолько мало, что вам не требуется заниматься длительным и болезненным предпродажным тестированием продукта перед тем, как вводить его в эксплуатацию;

- вам достаточно сформировать простой дизайн, который достаточен для текущей версии продукта, этот дизайн вовсе не обязан быть ориентированным на использование в течение всего времени жизни системы.

При выполнении этих условий вы, скорее всего, можете без опасений выпускать продукт небольшими версиями, внедряя его в полноценную эксплуатацию спустя короткое время после начала его разработки.

Метафора

Вы не можете приступать к разработке, обладая лишь метафорой. В метафоре недостаточно детально отображено строение системы, кроме того, что, если, формируя метафору, вы сделаете ошибку? Однако в XP:

- вы обладаете быстрой и надежной обратной связью, благодаря чему вы на основании реального кода и тестов узнаете о том, работает ли метафора на практике;
- вашему заказчику очень удобно разговаривать о системе в терминах метафоры;
- вы выполняете переработку (refactor) для того, чтобы постоянно пересматривать ваше представление о том, что метафора означает на практике.

При выполнении этих условий вы, скорее всего, можете без опасений приступать к разработке, обладая лишь метафорой.

Простой дизайн

Вы не можете разрабатывать систему, дизайн которой достаточен лишь для того, чтобы решать только сегодняшние задачи. Вы не можете проектировать систему, не уделяя внимание тому, с какими проблемами вам придется столкнуться завтра. Используя предлагаемый в XP подход, вы можете загнать себя в угол, при этом вы потеряете возможность дальнейшего эволюционирования системы. Однако в XP:

- вы привыкаете к постоянной переработке кода, благодаря чему внесение в дизайн системы не представляет для вас сложностей;
- вы обладаете понятной всеобщей метафорой, благодаря чему вы можете быть уверенными в том, что любые изменения в дизайне будут выполняться в рамках единого пути, сходящегося в одной точке;
- вы программируете не один, а с партнером, поэтому вы можете быть уверенными в том, что вы формируете простой дизайн, а не глупый дизайн.

При выполнении этих условий вы, скорее всего, можете без опасений строить систему на основе простого дизайна, который подходит для решения только лишь сегодняшних задач и не учитывает потенциальных проблем, которые могут возникнуть перед вами завтра.

Тестирование

Вы не сможете написать все эти тесты. Для этого потребуется слишком много времени. Мало того, программисты не любят писать тесты для программ, они любят разрабатывать сами программы. Однако в XP:

- дизайн системы настолько прост, насколько это вообще возможно, поэтому разработка тестов — это не такая уж сложная процедура;
- вы программируете не один, а вместе с партнером, поэтому если у вас нет возможности придумать еще один тест, этим может заняться ваш партнер, а если партнеру кажется, что он может нарушить работу теста, вы можете просто передать ему клавиатуру;
- вам становится приятно, когда вы видите, что все тесты работают;
- заказчикам становится приятно, когда они принимают в эксплуатацию систему, в отношении которой срабатывают все разработанные ими тесты.

При выполнении этих условий программисты и заказчики, скорее всего, будут тратить время на разработку тестов. Мало того, если вы не будете использовать автоматические тесты, все остальные составные части XP не будут работать столь же хорошо, как они должны работать.

Переработка кода

Вы не можете постоянно перерабатывать дизайн системы. Для этого потребуется слишком много времени, переработку будет сложно контролировать, кроме того, в процессе переработки вы с большой долей вероятности можете серьезно нарушить работу системы. Однако в XP:

- используется модель коллективного владения кода, поэтому вас не настораживает необходимость внесения в систему изменений тогда, когда в этом возникает необходимость;
- используются стандарты кодирования, благодаря чему вам не приходится реформатировать код перед тем, как осуществлять переработку;
- вы программируете парами, поэтому вам не страшно приступать к переработке кода в одиночку, кроме того, благодаря программированию в парах вероятность того, что вы нарушите работоспособность системы, снижается;

- вы используете простой дизайн, поэтому переработка кода выполняется проще;
- у вас есть тесты, поэтому вряд ли вы нарушите целостность кода, не узнав об этом сразу же;
- вы постоянно выполняете интеграцию системы, поэтому если вы нечаянно нарушите что-либо в системе или выполненная вами переработка конфликтует с чьей-либо работой, вы узнаете об этом в течение ближайших нескольких часов;
- вы полноценно отдыхаете после работы, поэтому у вас достаточно сил и храбрости для выполнения переработки, кроме того, вы реже совершаете ошибки.

При выполнении этих условий вы, скорее всего, можете без опасений перерабатывать код тогда, когда вы видите в этом необходимость. Необходимость в переработке возникает тогда, когда вы хотите сделать систему проще, устранить дублирование кода или обеспечить лучшую коммуникацию.

Программирование в парах

Вы не можете программировать парами. Двум людям очень сложно писать один и тот же код. Это слишком медленно. Кроме того, вам кажется, что два человека по отдельности на двух разных компьютерах напишут в два раза больше кода, чем вместе на одном компьютере. Однако в XP:

- стандарты кодирования снижают трения между членами пары;
- каждый из членов пары полноценно отдыхает, поэтому снижается вероятность бесполезных... э-э-э... дискуссий;
- программисты в парах разрабатывают тесты совместно, поэтому перед тем, как переходить к реализации, они сначала согласуют между собой свои представления о решении стоящей перед ними задачи;
- пары используют метафору для формирования таких решений, как именование элементов системы и базовый дизайн;
- пары работают в рамках простого дизайна, поэтому оба программиста в паре без затруднений понимают, что, собственно, происходит.

При выполнении этих условий вы, скорее всего, можете писать весь код приложения в парах вплоть до внедрения этого приложения в реальных производственных условиях. Мало того, если люди программируют в одиночестве, они с большей вероятностью совершают ошибки, они зачастую чрезмерно усложняют дизайн и действуют с нарушением других принятых методик работы (часто это происходит под внешним давлением).

Коллективное владение

Вы не можете разрешить каждому члену команды вносить любые изменения в любое место системы в любое удобное для этого время. В результате этого программисты немедленно разбросают код направо и налево и стоимость интеграции стремительно вырастет. Однако в XP:

- интеграция выполняется очень часто, каждые несколько часов, поэтому вероятность конфликтов снижается;
- вы пишете и постоянно запускаете тесты, поэтому вероятность нарушения работы системы снижается;
- вы программируете в парах, поэтому вероятность ошибок снижается и программисты понимают код быстрее, чем могут его изменить;
- вы действуете в рамках стандартов кодирования, поэтому конфликты стиля не возникают, а код становится понятным каждому члену команды (например, ситуация, когда разные люди привыкли ставить фигурные скобки по-разному, называется Curly Brace Wars — война фигурных скобок).

При выполнении этих условий вы, скорее всего, можете без опасений разрешить каждому члену команды изменять код в любой части системы тогда, когда ему это надо, и так, как он считает нужным для того, чтобы улучшить его. Мало того, если не использовать модель коллективного владения кодом, скорость эволюционирования дизайна значительно понижается, а это не идет на пользу системе.

Постоянно продолжающаяся интеграция

Вы не можете интегрировать всю систему каждые несколько часов работы. Интеграция — это очень серьезный и длительный процесс, в ходе которого, как правило, приходится устранять множество конфликтов, при этом может нарушиться целостность кода. Однако в XP:

- вы можете запустить тесты и быстро убедиться в том, что все работает;
- вы программируете парами, поэтому у вас наполовину меньше источников кода, которые требуется интегрировать и согласовывать между собой;
- вы выполняете переработку кода, и в процессе этого устраняется значительная часть конфликтов.

При выполнении этих условий вы, скорее всего, можете выполнять интеграцию каждые несколько часов. Мало того, если вы не выполняете интеграцию с достаточной частотой, вероятность возникновения конфликтов, равно как и стоимость интеграции, стремительно растет.

40-часовая рабочая неделя

Вы не можете работать только 40 часов в неделю. Вы не успеете выполнить весь необходимый объем работ. Однако в XP:

- игра в планирование заставляет вас делать наиболее важную работу в первую очередь;
- комбинация игры в планирование и тестирования снижает вероятность возникновения неприятных сюрпризов (именно такие сюрпризы являются причиной того, что приходится работать сверхурочно);
- весь набор рассматриваемых методик помогает вам программировать с максимальной скоростью, поэтому вы все равно не сможете сделать запланированный на неделю объем работ быстрее.

При выполнении этих условий вы, скорее всего, выполните весь запланированный на неделю объем работ в течение 40-часовой рабочей недели. Мало того, если команда не получит полноценного отдыха, люди будут сильно уставать и не смогут придерживаться всех остальных методик.

Заказчик на месте разработки

Вы не можете получить реального представителя заказчика, который все свое рабочее время сидел бы в офисе, где осуществляется разработка. Такой человек более ценен на производстве — там он приносит больше пользы для бизнеса. Однако в XP:

- представитель заказчика приносит пользу для проекта, так как он пишет функциональные тесты (предприятию-заказчику все равно придется этим заниматься);
- представитель заказчика приносит пользу проекту, устанавливая мелкомасштабные приоритеты и участвуя в принятии некоторых важных решений для программистов.

При выполнении этих условий представители заказчика смогут быть более полезными для компании благодаря тому, что они вкладывают свои усилия в разработку проекта. Мало того, если в состав команды не включить представителя заказчика, риск, связанный с проектом, увеличивается, так как программисты пытаются заранее предугадать, с какими задачами им придется иметь дело в будущем, они вынуждены заранее планировать структуру системы и кодировать, не зная при этом, какие тесты им придется удовлетворить, а какие тесты им разрешается игнорировать.

Стандарты кодирования

Вы не можете заставить команду кодировать в соответствии с общими для всех стандартами. Каждый программист — это индивидуальность, ему легче отказаться от общих стандартов, чем подчиниться требованию ставить фигурные скобки так, как делают это остальные члены команды. Однако в XP вся дисциплина XP способствует тому, что программисты в большей степени чувствуют себя членами команды, которая побеждает.

При выполнении этого условия входящие в команду программисты с большей охотой поправят свой стиль. Мало того, без использования стандартов кодирования дополнительные трения станут причиной значительного замедления программирования в парах и переработки кода.

Заключение

Любая из рассмотренных методик сама по себе плохо приспособлена для использования в рамках программистского проекта (за исключением, возможно, тестирования). Чтобы обеспечить их эффективное применение на практике, вы должны использовать их все одновременно. На рис. 4 изображена диаграмма, которая иллюстрирует взаимодействие всех методик. Линия между двумя практиками указывает на то, что обе эти практики являются поддержкой друг для друга. Я не хотел размещать этот рисунок в самом начале главы, так как при взгляде на него вы могли решить, что XP — это слишком сложная штука. Отдельные части сами по себе очень просты. Насыщенность и богатство происходят от взаимодействия между составными частями этой дисциплины.

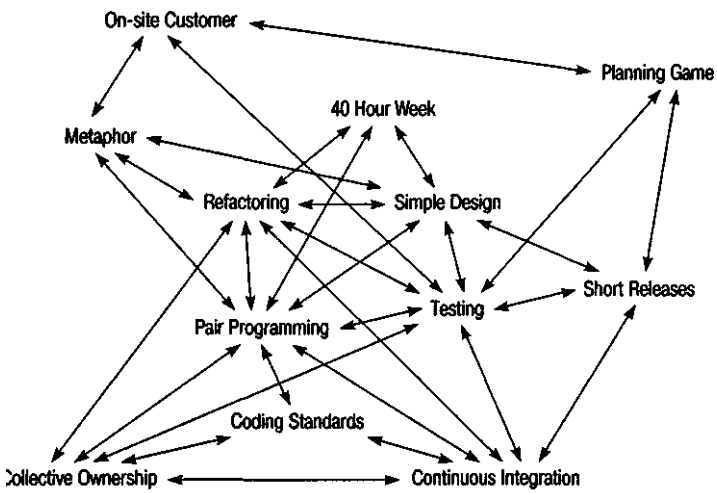


Рис. 4. Диаграмма, иллюстрирующая взаимодействие всех методик

Стратегия менеджмента 12

Мы будем осуществлять управление всем проектом, используя при этом базовые приемы бизнес-менеджмента, — поэтапная поставка, быстрая и надежная обратная связь, ясная взаимосвязь бизнес-требований к системе, а также использование специалистов для решения специализированных задач.

Дилемма менеджмента состоит в следующем. С одной стороны, было бы неплохо, если бы все решения принимал менеджер. При этом не приходится тратить ресурсы на коммуникацию, так как существует только один человек. Существует один человек, который отвечает за высший менеджмент. Существует один человек, который обладает видением. Никому больше не надо ничего знать, так как все решения принимает только один человек.

Мы знаем, что эта стратегия не срабатывает, так как ни один человек в мире не знает достаточно для того, чтобы принимать качественные решения во всех областях. Стратегии управления, сбалансированные в сторону централизованного контроля, также сложно реализовывать, так как при этом возникает чрезмерная нагрузка на тех, в отношении которых выполняется управление.

С другой стороны, прямо противоположная стратегия также не срабатывает. Вы не можете позволить кому угодно делать, что ему вздумается, без какого-либо надзора. Люди начнут действовать в разнобой. Требуется кто-то, кто обладал бы более общим взглядом на проект и мог бы повлиять на его развитие в случае, если работа команды отклоняется от намеченного курса.

И вновь мы должны вернуться к принципам, которые помогут нам проложить путь между двумя крайностями.

- Принимаемая ответственность, — в соответствии с этим принципом обязанность менеджера состоит не в том, чтобы назначить ту или иную работу тому или иному человеку, а в том, чтобы указать команде на необходимость выполнения этой работы.
- Качественная работа — предполагает, что взаимодействие между менеджерами и программистами должно основываться на доверии, так как программисты хотят делать свою работу хорошо. С другой стороны, это не означает, что менеджер может вообще ничего не делать. Однако необходимо понимать разницу между фразами: «Я стараюсь заставить этих ребят делать свою работу хорошо» и «Я должен помочь этим ребятам делать свою работу еще лучше».
- Постепенное изменение — предполагает, что менеджеры не выдают программистам в самом начале работы над проектом толстое описание внутренней политики, в рамках которой следует выполнять работу, а управляют процессом разработки в течение всего времени работы над проектом.
- Локальная адаптация — предполагает, что менеджеры должны взять на себя адаптацию XP к локальным условиям, анализируя моменты, в которых культура XP конфликтует с культурой компании, и формируя решения проблем в случае несоответствия.
- Путешествие налегке — предполагает, что менеджер не создает чрезмерной лишней нагрузки на проект (например, длительные совещания с обязательным присутствием всех членов команды или бесконечные продолжительные доклады о текущем состоянии проекта). Любая просьба, которую менеджер может адресовать программисту, должна требовать не очень много времени для исполнения.
- Откровенные оценки — предполагает, что любые метрики, собираемые менеджером, должны находится на реалистичном уровне точности. Не пытайтесь определять время с точностью до секунды, если у ваших часов есть только часовая и минутная стрелки.

Стратегия, которую можно сформировать на основании всего перечисленного, ближе к децентрализованному принятию решений, чем к централизованному контролю. Работа менеджера — вести игру в планирование (Planing Game) для того, чтобы собирать метрики, делать так, чтобы эти метрики были наблюдаемыми для тех, чья работа подвергается анализу и оценке, и время от времени вмешиваться в ситуации, в которых принятие решения распределенным образом невозможно.

Метрики

Основным инструментом оценки в XP является метрика. Например, отношение ожидаемого времени разработки к календарному времени является базовой мерой оценки в ходе игры в планирование. Эта мера позволяет команде установить *скорость проекта* (Project Velocity). Если отношение увеличивается (меньшее количество календарного времени для заданного ожидаемого объема разработки), это означает, что работа команды продвигается успешно. Или это может означать, что команда не делает ничего, кроме удовлетворения требований; подобная, с виду вполне благополучная ситуация может стать причиной возникновения проблем в дальней перспективе.

Средой отражения состояния метрик является *большая наглядная диаграмма* (Big Visible Chart). Вместо того чтобы посылать всем членам команды электронное письмо (которое зачастую будет ими игнорироваться), менеджер должен периодически (не реже чем раз в неделю) обновлять видимую для всех диаграмму. Зачастую подобное вмешательство просто необходимо. Вы полагаете, что написано недостаточное количество тестов? Покажите это на диаграмме и обновляйте этот показатель каждый день.

Не следует включать в состав диаграммы чрезмерно большое количество метрик. Будьте готовы убрать с диаграммы метрики, которые уже отслужили свою службу. Как правило, команда в состоянии следить одновременно за тремя-четырьмя показателями.

Со временем метрики теряют актуальность. На практике *любая* метрика, значение которой приблизилось к отметке 100%, перестает быть полезной. Это замечание, конечно же, не относится к показателю тестов модулей. Этот показатель всегда должен равняться 100%. Однако показатель тестов модулей — в большей степени предположение, а не метрика. Вы не можете отметить на диаграмме, что показатель функциональных тестов равен 97%, имея в виду, что осталось приложить усилия в размере 3%. Если значение метрики приближается к 100%, замените ее другой метрикой, которая, по вашему мнению, снизилась на несколько пунктов.

Все это не означает, что вы можете управлять проектом XP «при помощи чисел». Напротив, числа в данном случае — это способ мягко и непринужденно сообщить людям о необходимости изменений. Для менеджера XP наиболее чувствительным барометром необходимости изменений является внимательное слежение за его собственными ощущениями: физическими и эмоциональными. Если утром, когда вы садитесь в свою машину, ваш желудок завязывается узлом, это значит, что с вашим проектом происходит что-то неправильное и ваша работа состоит в том, чтобы что-то изменить.

Инструктирование

То, что многие понимают под менеджментом, в XP делится на две роли: *инструктор* (coach) и *ревизор* (tracker). Обе эти роли могут исполняться одним и тем же членом команды, однако, безусловно, это могут быть также разные люди. Инструктирование в основном связано с техническим выполнением (и эволюцией) процесса. Идеальный инструктор должен обладать хорошими навыками общения, он не должен легко поддаваться панике, должен обладать хорошими техническими навыками (однако это не абсолютно необходимое требование) и должен быть уверенным в себе. Зачастую возникает желание использовать в качестве инструктора человека, который в других командах выполнял бы функции ведущего программиста или системного архитектора. Однако роль инструктора в XP существенно отличается.

Термины «ведущий программист» или «системный архитектор» рождают в голове видение изолированного гения, который принимает важные решения о проекте. Инструктор XP — это прямо противоположное понятие. Инструктор XP тем лучше, чем меньше он делает технических решений: его работа состоит в том, чтобы помогать другим людям принимать хорошие решения.

Инструктор не принимает на себя ответственность за множество технических задач. Скорее, его обязанности заключаются в следующем:

- быть доступным в качестве партнера по разработке, особенно для новичков, которые только начинают брать на себя ответственность за решение сложных технических задач;
- видеть долгосрочные цели переработки кода и стимулировать мелкомасштабную переработку для того, чтобы частично способствовать достижению этих целей;
- помогать программистам индивидуальными техническими навыками, например тестированием, форматированием и переработкой;
- объяснять процесс разработки менеджерам высшего звена.

Однако, наверное, самой главной обязанностью инструктора является покупка игрушек и еды. Проекты XP похоже притягивают к себе игрушки. Консультанты часто рекомендуют использовать обычные головоломки для стимулирования побочных мыслительных процессов. Однако в XP игрушка необязательно должна быть головоломкой. Инструктор использует игрушки для того, чтобы глубоко и продуктивно воздействовать на процесс разработки. Например, во время работы над проектом Chrysler C3 совещания о проектировании зачастую тянулись в течение многих часов и их участники все никак не могли прийти к приемлемому решению.

Чтобы решить проблему, я купил обычный кухонный будильник и заявил, что ни одно совещание не может тянуться более 10 минут. Я не уверен в том, что этот будильник был когда-либо использован по прямому назначению, однако его видимое присутствие напоминало всем о том, что надо внимательно следить за ходом дискуссии. Если обсуждение переставало быть результативным, все глядели на будильник и понимали, что пора пойти и написать какой-либо код для того, чтобы проверить свои доводы на деле.

Еда также является отличительным признаком всех проектов XP. Есть нечто значительное в том, что вы преломляете хлеб в компании со своими соратниками. Если во время разговора вы что-то жуете, дискуссия начинает идти совсем по-другому. Поэтому в XP-проектах еда постоянно разложена повсюду. Лично я рекомендую шоколадные батончики «Frigor Noir», если конечно, вы сможете их раздобыть, однако я наблюдал, что некоторые проекты вполне сносно выживают на палочках из лакрицы «Twizzler». Полагаю, что вы в состоянии разработать свое собственное локальное меню.

Роб Мии (Rob Mee) пишет.

> Эти наборы тестов очень коварны. В моей команде мы практикуем вознаграждение самих себя едой и питьем за успешно завершенное тестирование. Например, в 14:45 я говорю: «Если мы закончим с этим до 15:00, мы сможем перекусить и выпить чаю». Конечно же, мы сможем перекусить в любом случае, даже если тестирование затянется вплоть до 15:15. Однако мы не будем есть до тех пор, пока все тесты не сработают, — если у нас есть задача и цель, к которой мы стремимся, то перекус, который мы осуществляем, достигнув цели, превращается в маленькое празднество.

Слежение

Слежение (tracking), или, точнее говоря, *отслеживание,* — это еще один важный компонент менеджмента в XP. Вы можете делать любые предположения и оценки, но если вы не будете следить за тем, как дела идут в действительности и насколько действительность отличается от ваших предположений, вы не сможете научиться делать правильные оценки.

Слежение за ходом дел в XP осуществляет *ревизор* (tracker). Ревизор собирает сведения о состоянии метрик, которые отслеживаются в данный момент, кроме того, он следит за тем, чтобы все члены команды знали о том, какие метрики в данный момент отслеживаются (также он напоминает о ранее сделанных предположениях и оценках).

Ведение игры в планирование — это часть слежения. Ревизор должен отлично знать правила игры и быть готовым применить их в различных эмоциональных ситуациях (планирование всегда эмоционально).

Слежение должно осуществляться без лишних трудозатрат и неудобств. Если ответственное лицо по два раза на дню будет спрашивать у программи-

стов о текущем состоянии их дел, в скором времени программисты начнут уклоняться от такого контроля. Напротив, ревизор должен экспериментировать, чтобы выяснить, какой минимальный объем измерений необходимо выполнить для того, чтобы при этом не утратить эффективности и актуальности этих измерений. Сбор реальных данных о разработке дважды в неделю — этого вполне достаточно. При более интенсивных измерениях вы вряд ли получите лучшие результаты.

Интервенция

Управление командой XP — это не только хождение за едой и покупка игрушек. В некоторых ситуациях возникшие проблемы просто не могут быть решены за счет гениальности независимой и самостоятельно действующей команды, опекаемой любящим и бдительным менеджером. В подобных ситуациях менеджер XP должен быть в состоянии выполнить свою основную функцию. Иными словами, он должен быть готовым принимать важные решения — даже самые непопулярные — и наблюдать за последствиями этих решений до самого конца. Вмешательство менеджера — это и есть интервенция.

Прежде всего менеджер должен внимательно проанализировать ситуацию, пытаясь понять, существовало ли что-либо, о чем необходимо было знать ранее или что необходимо было бы сделать ранее для того, чтобы полностью избежать возникновения проблемы. Время, когда требуется вмешательство менеджера, это не время надевать белые доспехи и вскакивать на боевого коня. Скорее, это время, когда надо прийти к команде и сказать: «Я не могу понять, как я позволил этому произойти, но сейчас я вынужден сделать то-то и то-то». Смирение — основное правило для интервенции.

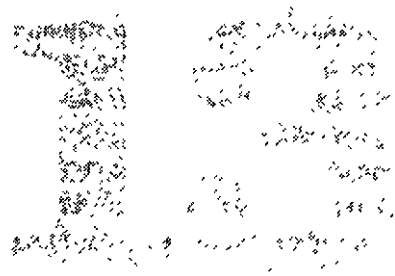
Одной из причин, достаточно серьезных для того, чтобы вмешаться в процесс разработки, является изменение в составе команды. Если член команды не справляется со своими обязанностями, менеджер вынужден попросить его уйти. Подобные решения лучше принимать раньше, чем позже. Как только вы не можете представить себе ситуацию, в которой изучаемый вами субъект мог бы оказать поддержку и не был бы обузой, вы должны сделать свой ход. Дальнейшее ожидание только лишь усугубит проблему.

Несколько более приятной обязанностью менеджера является вмешательство в момент, когда работа команды требует изменений. Как правило, менеджер не должен приказывать, что именно требует изменений и какими должны быть эти изменения. Менеджер должен всего лишь указать на необходимость изменений. Команда должна придумать и провести один

или несколько экспериментов, после чего менеджер докладывает команде о том, как изменились показатели в результате проведения эксперимента.

Наконец, еще одной причиной для интервенции является закрытие проекта. В большинстве ситуаций команда не сможет отказаться от продолжения работы над проектом по собственной воле. Однако приходит день, когда дальнейшие инвестиции в текущую систему становятся менее привлекательными, чем какая-либо другая альтернатива, например начало работы над новой системой, которая должна прийти на смену устаревшей. Менеджер обязан быть в курсе, когда именно происходит переход через эту пограничную черту. Он также обязан известить более высокое руководство о необходимости изменений.

Стратегия организации рабочего места



Мы создадим для нашей команды открытое рабочее место с небольшими индивидуальными пространствами по периферии и общей областью программирования в середине.

Рон Джеффрис (Ron Jeffries) писал о фотографии, показанной на рис. 5.

На этой фотографии показано рабочее помещение команды DaimlerChrysler C3, работающей над проектом Payroll. В помещении находятся два больших стола, на каждом из которых установлено по шесть программистских компьютеров. Пары программистов располагаются за любыми доступными для этого компьютерами. На картинке ничего не срежиссировано специально: программисты действительно работают за одним компьютером в паре, именно так, как показано. Тот, кто фотографировал, на самом деле работает в паре с Четом — программистом, который сидит за дальним столом спиной к фотокамере.

На двух видимых на фотографии стенах размещаются белые доски, на которых показаны функциональные тесты, требующие внимания, запланированные сеансы CRC, а также план итераций на задней доске. Листки бумаги, которые можно заметить в верхней части доски слева, — это небольшие заметки, в которых отмечены используемые группой правила XP.

Вдоль правой стены располагаются небольшие отгороженные от остального помещения индивидуальные ячейки, в которых можно поставить телефон или использовать их для того, чтобы написать что-нибудь на бумаге, не отвлекаясь на других людей.

В дальней части помещения, между компьютерным столом и белой доской, располагается стандартный стол, вокруг которого команда собирается для CRC-сеансов. Как правило, стол завален карточками CRC и едой (одно из используемых командой правил гласит: «здесь должна быть еда»).

Помещение было спроектировано командой: мы действительно РЕШИЛИ работать в этом помещении. Люди говорят негромко, поэтому уровень шума на удивление низкий. Однако если вы нуждаетесь в помощи, вы можете немножко повысить голос, чтобы получить эту помощь. Помощь придет незамедлительно: обратите внимание, что на полу отсутствует ковер. Это значит, что стулья, на которых мы сидим, могут перемещаться по комнате без каких-либо затруднений.

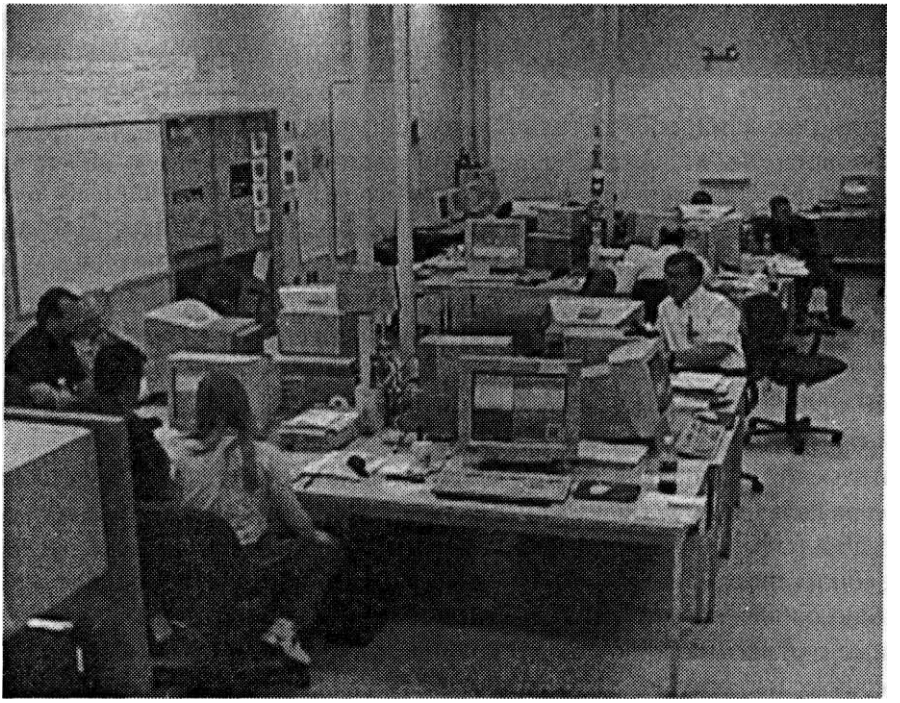


Рис. 5. Рабочее помещение команды DaimlerChrysler C3

Если у вас нет подходящего места для работы, ваш проект не сможет стать успешным. Различие между хорошим местом для команды и плохим местом для команды значительно и подчас критично.

В моей карьере консультанта был момент, когда я особенно остро ощутил значимость правильной организации рабочего места. Я был приглашен в одну организацию для того, чтобы пересмотреть объектно-ориентированный дизайн для некоторого проекта. Я взглянул на систему и, конечно же, обнаружил, что она была спроектирована из рук вон плохо. После этого я обратил внимание на то, кто где сидит. Команда состояла из четырех старших программистов. Каждый из них работал в своем собственном кабинете, а каждый из этих кабинетов располагался в одном из четырех углов здания средних размеров. Я посоветовал им переселиться так, чтобы их кабинеты располагались рядом друг с другом. Я был приглашен из-за того, что обладал значительным опытом в области Smalltalk и объектно-ориентированного дизайна, однако, на мой взгляд, наиболее ценным советом, который я им дал, было предложение переставить мебель.

Правильная организация рабочих помещений — это в любом случае непростая работа. В этой области существует большое количество конфликтующих ограничений. Планировщики рабочих помещений стараются

потратить как можно меньше денег и при этом обеспечить как можно большую гибкость. Люди, использующие эти помещения, желают работать в тесной связке с остальной командой. В то же время они нуждаются в некотором отделенном от остальных людей индивидуальном пространстве, из которого они могли бы (например) звонить в приемную своей поликлиники.

В рамках XP существует сильная склонность к огромным публичным пространствам. XP — это коммунальная дисциплина разработки программного обеспечения. Члены команды должны видеть друг друга, слышать изредка выкрикиваемые вопросы, «случайно» подслушивать сторонние переговоры, в которые они могли бы внести жизненно важные замечания.

XP преображает структуру рабочих помещений. Общепринятая структура офисов плохо соответствует идеологии XP. Например, если вы поставите свой компьютер в угол, это будет не самым удачным решением, так как двум людям будет неудобно работать с таким компьютером. Часто рабочее помещение — это огромный зал, поделенный перегородками на небольшие индивидуальные рабочие ячейки, однако такая структура тоже не работает в рамках XP. Для повышения эффективности XP необходимо снизить высоту стен по крайней мере на половину, а лучше вообще от них избавиться. В то же время команда должна быть отделена от остальных команд.

Лучшим является план помещения, в котором центральное обширное общее пространство обрамлено по периметру небольшими отгороженными ячейками. Члены команды могут хранить в этих ячейках свои личные вещи, делать в них телефонные звонки, или проводить в них время в одиночестве, когда они не хотят, чтобы их отвлекали. Вся остальная команда должна с уважением относиться к «виртуальному» одиночеству человека, сидящего в собственной ячейке. Самые большие, самые быстрые компьютеры следует разместить в середине центрального общего пространства (индивидуальные ячейки могут быть как с компьютерами, так и без компьютеров). В этом случае, если кто-то хочет приступить к программированию, он будет вынужден выйти на открытое общее пространство. Здесь каждый может увидеть, что происходит; формирование пар осуществляется без затруднений и каждая пара получает дополнительную энергетику от других пар, которые также занимаются программированием поблизости.

Если у вас есть возможность, вы можете зарезервировать наиболее уютный кусочек помещения рядом с общим рабочим пространством, разместить там кофейный комбайн, пирожки, несколько игрушек, нечто, что будет притягивать туда людей. Очень часто для того, чтобы выйти из тупика, который возник в процессе разработки, достаточно отойти ненадолго

в сторону и отвлечься. Если при этом у команды будет специально предназначенное место, люди с большей вероятностью воспользуются этим в случае необходимости.

Одна из рассмотренных ранее ценностей — храбрость — находит свое воплощение в тяге XP к рабочим помещениям. Если корпоративный взгляд на планировку рабочего пространства конфликтует со взглядом команды, команда, как правило, побеждает. Если компьютеры размещены в неправильных местах, они перемещаются в новые места. Если помещение разделено перегородками неправильно, перегородки исчезают. Если освещение слишком яркое, оно приглушается. Если телефоны звонят слишком громко, однажды обнаруживается, что в звонке каждого из них набита вата.

Однажды мы работали для банка, и когда я в самый первый день пришел на работу, я обнаружил, что для работы нам выделили отвратительные старые столы, за каждым из которых мог разместиться только один человек. По обе стороны от каждого стола были смонтированы металлические выдвижные ящики, таким образом, столы даже нельзя было сдвинуть вместе. Между выдвижными ящиками умещались ноги только одного человека. Очевидно, что с этим невозможно было работать. Мы посмотрели вокруг и обнаружили электрическую отвертку с достаточно мощным двигателем. После этого мы отвинтили у одного из столов один набор выдвижных ящиков. В результате за этим столом теперь могли разместиться не один, а два человека. Аналогичным образом мы обработали все остальные столы.

Вся подобная возня с мебелью может стать для вас причиной дополнительных административных проблем. Люди, в обязанности которых входит следить за состоянием рабочих помещений и инвентаря, могут прийти в негодование от того, что кто-то без их ведома передвигает столы (не думая о том, что если обратиться к ним с этой просьбой, то ее исполнение может потребовать несколько недель или даже месяцев). На это я отвечу: «Очень плохо». Я должен разработать программную систему, и если я избавлюсь от этих дурацких перегородок, я смогу справиться с моей задачей лучше и быстрее, поэтому я иду и убираю то, что мешает мне работать. Если организация, для которой я работаю, не может вытерпеть столь смелую инициативу, значит, я просто не хочу для нее работать.

Если вам удается установить контроль над физическим окружением, это значит, что ваша команда получает чрезвычайно важное сообщение: никто не собирается смиряться с какими бы то ни было иррациональными интересами организации, которые встают на пути к успеху. Установление контроля над физической средой — это первый шаг в направлении установления контроля над общим стилем работы команды.

Мебель и рабочие помещения могут быть предметом постоянного экспериментирования (еще одна ценность — обратная связь — в действии). В конце концов, организация потратила кучу денег на приобретение всего этого гибкого офисного инвентаря. Все эти деньги будут потрачены впустую, если вы не воспользуетесь этой гибкостью и не перенастроите свой офис так, как вам это нужно. Что, если сделать индивидуальные ячейки ближе друг к другу? А что, если их немножко удалить? Может, поставить компьютер, предназначенный для интеграции, в центре зала? А может, лучше в углу? Попробуйте это. То, что оказывается удобным, остается. Тем, что не работает, жертвуют в процессе следующего эксперимента.

14 Разделение полномочий между технарями и бизнесменами

Одним из основополагающих правил нашей стратегии является то, что технари концентрируются на решении технических **проблем,** а бизнесмены — на решении бизнес-проблем. Проект должен управляться **бизнес-решениями,** однако для принятия бизнес-решений должна использоваться информация о затратах и риске, предоставляемая **техническим** *л* специалистами. Эта информация является результатом технических решений.

Существуют два широко распространенных неправильных режима взаимоотношений между бизнесом и разработчиками: когда либо бизнес, либо разработчики получают слишком большую власть над проектом, проект начинает страдать.

Бизнес

Когда бизнесмены получают слишком много полномочий, они начинают диктовать разработчиком значения для всех четырех переменных. «Вот то, что ты должен сделать. Это должно быть сделано тогда-то и тогда-то. Нет, тебе не дадут ни одной дополнительной рабочей станции. И для тебя будет лучше, если ты сделаешь эту работу с наивысшим возможным качеством, иначе у тебя будут проблемы. Ты меня хорошо понял? Скотина ленивая!»

В такой ситуации бизнес предписывает слишком многое. Некоторые элементы в списке требований абсолютно обязательны, но некоторые — нет. И если у разработчиков не будет никаких полномочий, они не смогут возразить. Они не смогут принудить бизнес выбрать правильный вариант. И тогда разработчики, понурив голову, идут работать над невыполнимой задачей, которую перед ними поставили.

Как правило, наименее важные требования являются причиной наибольшего риска. Похоже, это является следствием их природы. Они меньше

всего обдумываются, меньше всего анализируются и меньше всего осмысливаются, поэтому вероятность того, что именно они изменятся в процессе разработки, выше всего. Очень часто такие требования оказываются также наиболее рискованными с технической точки зрения.

В результате, если бизнес получает слишком большие полномочия, проект требует слишком много усилий и генерирует слишком много риска, при этом он обеспечивает слишком незначительную отдачу.

Разработчики

Можно подумать, что если большие полномочия предоставляются разработчикам, жизнь становится лучше. Но на самом деле это не так. В действительности результат получается такой же.

Когда разработчикам предоставляется чрезмерная свобода, они начинают использовать все те новые технологии и процессы, для которых у них никогда не хватает времени, если «эти белые воротнички» постоянно подгоняют их. Когда разработчикам предоставляется свобода, они устанавливают и начинают использовать новые инструменты разработки, новые языки программирования, новые технологии. При этом инструменты, языки и технологии выбираются исходя из того, что они очень интересны и суперсовременны. Все только что появившееся на рынке связано с риском. (Если мы не попробуем это сейчас, то когда же еще?)

Таким образом, в результате предоставления разработчикам слишком широких полномочий, они прикладывают слишком много усилий и генерируют слишком много риска, при этом они обеспечивают слишком незначительную отдачу.

Что делать?

Решение состоит в том, чтобы определенным образом разделить полномочия и ответственность между бизнесом и разработчиками. Бизнесмены должны принимать решения в своей области компетенции, а программисты должны принимать решения в своей области компетенции. Решения, принятые одной стороной, должны стать базой для решений, принимаемых другой стороной. Ни одна сторона не должна в одностороннем порядке решать абсолютно все.

Обеспечение подобного политического баланса может показаться невозможным. Если ООН не в состоянии добиться этого, то какие могут быть шансы у вас? Конечно же, если все, что у вас есть, — это туманная цель «достижения баланса политических сил», тогда вы действительно остаетесь без шансов. Первая же сильная личность, которая появится на сцене, станет причиной нарушения баланса. К счастью, цель может быть гораздо более определенной, чем упомянутая.

Для начала небольшое отклонение от темы. Если кто-нибудь спросит у меня, какой автомобиль я хочу: «Феррари» или минивэн, я уверен, что выберу «Феррари». Но если этот кто-то спросит у меня: «Хочешь ли ты „Феррари" за 200 000 франков или минивэн за 40 000?» — я начну формировать взвешенное решение. Если добавить к этому еще пару требований, например, «мне необходимо брать с собой в дорогу пять детей» или «я должен быть способен развить скорость 200 километров в час», картина еще более прояснится. Существуют случаи, в которых каждое из решений по-своему оправданно, однако вы, скорее всего, не сможете сформировать хорошее решение исходя только лишь из глянцевых фотографий. Вы должны знать, какими ресурсами вы обладаете, в чем вы ограничены и во что вам обойдется каждый из вариантов.

Следуя этой модели, бизнесмены должны определить:

- объем работ или время выпуска версий продукта;
- относительные приоритеты предполагаемых возможностей системы;
- конкретный объем работ, связанных с предполагаемыми возможностями системы.

Для того чтобы помочь бизнесменам сформировать эти решения, разработчики должны сообщить:

- Оценки времени, необходимого для реализации различных возможностей системы.
- Предположительные последствия использования различных технических альтернатив.
- Процесс разработки, который соответствует их индивидуальности, их бизнес-окружению и культуре компании. Не существует списка пунктов типа: «Вот так мы пишем программное обеспечение...», который бы подходил для абсолютно любой ситуации. Все это потому, что ситуация постоянно меняется.
- С использования какого набора методик они начнут свою работу и при помощи какого процесса они будут пересматривать эффекты использования этих методик и экспериментировать с изменениями. Это напоминает Американскую конституцию, которая устанавливает базовую философию, базовый набор правил (билль о правах и первые 10 поправок), а также правила изменения правил (добавления новых поправок).

Так как бизнес-решения формируются в течение всего времени жизни проекта, назначение бизнесменам ответственности за принятие бизнес-решений подразумевает, что заказчик является таким же членом коман-

ды разработчиков, как и любой другой программист. На практике для получения лучших результатов заказчик в течение всего рабочего времени сидит вместе с остальной командой и отвечает на возникающие у программистов вопросы.

Выбор технологии

Поначалу может показаться, что выбор технологии — это чисто техническое решение, но на самом деле это бизнес-решение, однако бизнесмены должны формировать его на основе сведений, предоставляемых разработчиками. Заказчик будет вынужден взаимодействовать с поставщиком базы данных или языка программирования в течение многих лет, поэтому он должен быть уверен в хороших с ним отношениях как на бизнес-уровне, так и на техническом уровне.

Если заказчик говорит мне: «Мы хотим использовать вот эту реляционную базу данных и вот эту среду Java IDE», — я должен рассказать ему о последствиях подобного решения. Если я подозреваю, что объектно-ориентированная база данных и C++ лучшим образом подходят для данного проекта, я выполню оценку проекта с двух точек зрения. После этого бизнесмены получат возможность сформировать взвешенное бизнес-решение.

Однако существует еще одна сторона технологических решений, которая имеет прямое отношение к технарям, а не к бизнесменам. Как только технология начинает использоваться в рабочей среде компании, кто-то должен заниматься ее обслуживанием и поддержкой ее функционирования. Затраты, связанные с использованием пусть даже самой новейшей и самой совершенной технологии, — это не только затраты, связанные с разработкой системы на базе этой технологии. Эти затраты включают в себя также стоимость технической поддержки и обслуживания, одним словом, поддержки жизнедеятельности разработанной системы.

Что если это сложно?

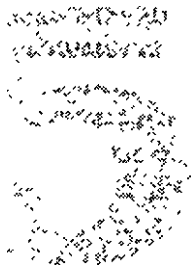
В большинстве случаев решения, которые формируются в рамках этого процесса, на удивление просто воплотить в жизнь. Программисты отлично видят чудовищ, которые притаились за каждой из историй. Бизнесмены часто говорят: «Я и не думал, что это будет настолько дорого. Давайте сделаем это же самое, только на треть. Я полагаю, что на текущий момент этого будет достаточно».

Однако в некоторых случаях подобный способ не срабатывает. В некоторых ситуациях наименьшая ценная часть разработки с точки зрения бизнеса выглядит большой и рискованной с точки зрения программиста. Когда такое происходит, вы не должны допускать каких-либо сомнений.

Вы обязаны действовать осторожно. Допускается сделать лишь незначительное количество ошибок. Вы можете потребовать большее количество внешних ресурсов, однако может случиться так, что вы получите эти ресурсы только тогда, когда времени будет в обрез. Поэтому с самого начала вы должны сделать все, что в ваших силах, для того, чтобы сократить объем работ. Вы должны сделать все, что в ваших силах для того, чтобы снизить риск. Когда все это будет сделано, вы можете приступать к работе.

Об этом можно сказать и по-другому: разделение полномочий между бизнесом и разработчиком — это не оправдание отказа от выполнения «грязной» работы. Совсем наоборот. Это способ отделения той работы, которая очевидно является сложной, от той работы, про которую вы просто еще не придумали, как сделать ее простой. В большинстве случаев работа оказывается проще, чем вам кажется с самого начала. Если это не так, вы обязаны сделать работу в любом случае, так как именно за это вам и платят деньги.

Стратегия планирования



Мы будем планировать нашу работу следующим образом: сначала мы быстро сформируем общий план, затем мы будем постоянно пересматривать его, формируя более конкретные цели для более коротких сроков: лет, месяцев, недель и дней. Мы будем формировать план быстро и с минимальными затратами, поэтому если нам потребуется изменить его, мы должны будем преодолеть минимальную инерцию.

Планирование — это процесс предположения, как будет выглядеть процесс разработки программного продукта для заказчика. Вот некоторые цели, которые достигаются в процессе планирования:

- собрать команду вместе;
- определить объем работ и приоритеты;
- оценить затраты и график работ;
- добиться появления у каждого заинтересованного лица ощущения того, что система действительно может быть реализована;
- определить исходные данные для обратной связи.

Давайте взглянем на принципы, которые влияют на планирование. Некоторые из них являются базовыми принципами, о которых шла речь в главе 8. Другие относятся конкретно к планированию.

- Планируйте только то, что вам нужно для реализации очередного этапа — на любом уровне детализации осуществляйте планирование только очередного этапа работ — то есть следующей версии, завершения следующей итерации. Это не означает, что вы не должны выполнять долгосрочное планирование. Вы можете это делать, только не с высокой степенью детализации. Вы можете спланировать текущую версию достаточно скрупулезно, и при этом вы можете

обрисовать план пяти следующих версий набором простых тезисов. При этом вам не придется тратить значительных усилий на то, чтобы спланировать все шесть версий с высокой степенью детализации.

- Принимаемая ответственность — ответственность может быть только принята, но не может быть назначена. Это означает, что в рамках XP не может быть такой вещи, как планирование сверху вниз. Менеджер не может прийти к команде и сказать: «Вот то, что мы должны сделать и это должно быть сделано за такое-то и такое-то время». Менеджер проекта должен предложить команде взять ответственность за выполнение работы. После этого он должен внимательно выслушать ответ.
- Предположительные оценки должны выполняться лицом, ответственным за реализацию, — если команда берет на себя ответственность за выполнение некоторой работы, она должна сообщить, какое время для этого потребуется. Если член команды берет на себя ответственность за решение некоторой задачи, он должен сообщить, сколько ему для этого потребуется времени.
- Игнорируйте взаимосвязи между частями проекта — планируйте так, как если бы части разработки можно было бы менять между собой по вашему собственному желанию. Это простое правило избавит вас от проблем при условии, что вы будете в первую очередь реализовывать наиболее высокоприоритетные бизнес-требования. «Сколько стоит кофе?» «25 центов за чашку, но вторая чашка бесплатно». «Тогда дайте мне вторую чашку». Подобные ситуации не должны происходить.
- Планируйте в соответствии с определенной целью: для определения приоритета или для осуществления разработки — не забывайте о том, зачем вы планируете. Если вы планируете для того, чтобы заказчик мог определить приоритеты, вы можете делать это с меньшей детализацией. Если же вы планируете для осуществления реализации, где вы должны проанализировать специфические тестовые случаи, вам потребуется существенно более высокий уровень детализации.

Игра в планирование

Планирование в XP намеренно абстрагирует процесс планирования для двух участников — бизнес (Business) и разработчики (Development). Это способствует устранению некоторого эмоционального напряжения, которое часто возникает в процессе обсуждения планов. Вместо «Джо, ты придурок! Ты же обещал мне сделать это еще к минувшей пятнице!» игра в планирование (Planning Game) сообщает: «Разработчики обнаружили

нечто. Им нужна помощь со стороны бизнеса для того, чтобы среагировать на открытие лучшим способом». Нельзя устранить эмоциональное напряжение при помощи простого набора правил, да мы и не собираемся этого делать. Правила существуют для того, чтобы напомнить каждому, как он должен действовать, также на правила можно сослаться в случае, если дела идут не так, как хотелось бы.

Зачастую бизнес не любит разработчиков. Отношения между людьми, которые нуждаются в системах, и людьми, которые эти системы разрабатывают, зачастую напоминают отношения между многовековыми врагами. Недоверие, взаимные обвинения, хитрое и скрытое маневрирование — все это вполне характерные вещи. Вы не можете разрабатывать хорошее программное обеспечение в подобных условиях.

Если упомянутые мною признаки не относятся к вашей рабочей среде, значит, вам повезло. Лучшей является рабочая среда, основанная на взаимном доверии. Каждая сторона уважает своего партнера. Каждая сторона уверена в том, что ее партнер сделает все от него зависящее для того, чтобы достичь поставленной цели. Каждая сторона желает помочь партнеру, вкладывая в общее дело собственные навыки, опыт и лучшие намерения.

Подобные взаимоотношения нельзя установить при помощи законов и инструкций. Вы не можете просто сказать: «Мы понимаем, что мы ведем себя несправедливо по отношению друг к другу. Нам ужасно жаль. Это больше не повторится. Давайте сразу же после обеда начнем сотрудничать совершенно по-другому». Весь мир и все люди не могут работать таким образом. В состоянии стресса люди начинают вести себя по-старому, как бы плохо это ни было.

Чтобы обеспечить отношения взаимного уважения, необходимо сформировать набор правил, которые определяли бы методику взаимодействия таким образом, чтобы было меньше поводов к ссорам. Кто именно должен принимать то или иное решение, когда это решение должно быть принято, в каком порядке осуществляется документирование решений.

Однако никогда не следует забывать, что правила игры — это всего лишь поддержка, шаг в сторону отношений, которые вы хотели бы установить с вашим заказчиком. Правила никогда не могут полноценно заменить утонченность, гибкость и чувственность реальных человеческих отношений. Все же без определенного набора правил вы никогда не сможете улучшить ситуацию. Когда у вас есть правила и ситуация начинает улучшаться, вы можете приступить к модификации правил для того, чтобы обеспечить более эффективную разработку. Со временем, когда правила превратятся в привычку, вы можете вообще забыть об их существовании.

Однако для начала вы должны научиться играть в соответствии с правилами. Вот они.

Цель

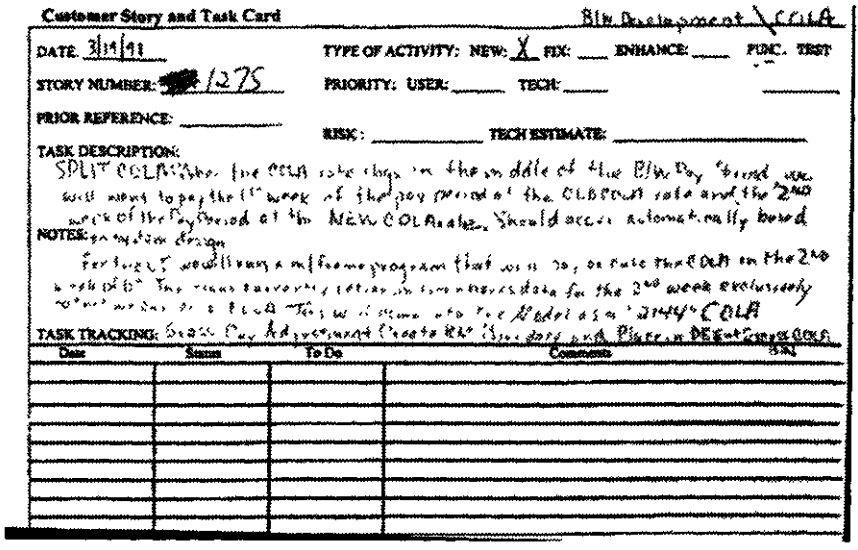
Цель игры — максимизировать ценность программного обеспечения, разработкой которого занимается команда. Исходя из ценности программы вы можете получить затраты, связанные с ее разработкой, а также уровень риска, которому вы подвергаетесь в процессе разработки.

Стратегия

Стратегия команды — инвестировать настолько мало, насколько это возможно для того, чтобы внедрить наиболее ценную функциональность в реальных производственных условиях так быстро, как только это возможно, но только в соответствии со стратегиями программирования и дизайна, направленными на снижение риска. В свете технологических и бизнес-выводов, которые формируются исходя из опыта разработки и эксплуатации этой самой первой системы, бизнес определяет, что теперь является наиболее ценной для него функциональностью, а команда как можно быстрее внедряет эту функциональность в реальных производственных условиях. И так далее.

Куски

Куски (pieces) в игре в планирование — это карточки историй (story cards). Пример одной из них показан на рис. 6.


Customer Story and Task Card

DATE 3/19/98 TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ PDIC: TEST

STORY NUMBER: [redacted]1275 PRIORITY: USER: ___ TECH: ___

PRIOR REFERENCE: ___ RISK: ___ TECH ESTIMATE: ___

TASK DESCRIPTION:
SPLIT COLA: When the COLA rate changes in the middle of the Biz Day Period we will need to pay the 1st week of the pay period at the OLD COLA rate and the 2nd week of the period at the NEW COLA rate. Should occur automatically based

NOTES: on system design

For target week, write a mainframe program that will go out and create the COLA on the 2nd week of the pay period for the newest data for the 2nd week exclusively

TASK TRACKING: Status Day Assessment: Create the User Story and Place in DEV-Complete

Date	Status	To Do	Comments	Init

Рис. 6. Карточка с историей (story card)

Игроки

В игру в планирование играют два игрока — *бизнес* (Business) и *разработчики* (Development). Разработчики — это люди, которые отвечают за реализацию системы. Бизнес — это люди, которые принимают решения о том, что должна делать система.

Иногда сразу ясно, кто именно должен играть на стороне бизнеса. Если компания — биржевой брокер платит деньги за разработку специализированного программного обеспечения, значит, эти люди играют на стороне бизнеса. Именно они должны определить, что необходимо сделать в первую очередь. Но если вы занимаетесь разработкой программного продукта, который предназначен для массового рынка? Кто тогда играет роль бизнеса? Бизнес должен принимать решения относительно объема работ, приоритетов и содержимого версий продукта. Все эти решения, как правило, делаются сотрудниками отдела маркетинга. Если это достаточно умные люди, они будут принимать решения исходя из мнения:

- реальных пользователей продукта;
- заинтересованных в продукте групп;
- людей, занимающихся продажами.

Лучше всего на стороне бизнеса играют пользователи-эксперты. Например, в свое время я работал над созданием системы обслуживания клиентов для финансового фонда взаимных вкладов. На стороне бизнеса играла женщина-руководитель отдела работы с клиентами, которая в свое время, в самом начале своей карьеры, в течение долгих лет работала в качестве обычного клерка со старой компьютерной системой. Она изучила ее во всех подробностях. Время от времени у нее возникали проблемы, связанные с тем, что она не могла отличить то, что должна делать новая система, от того, что делала старая. Однако после того, как она привыкла к составлению историй, она отлично освоилась.

Ходы

Игра состоит из трех фаз:

- *исследование* (exploration) — в этой фазе необходимо определить, что нового должна делать система;
- *подтверждение* (commitment) — в этой фазе необходимо решить, какое подмножество всех возможных требований необходимо удовлетворить в следующую очередь;
- *управление* (steer) — в этой фазе необходимо управлять разработкой по мере того, как реальность вносит свои коррективы в план.

Ходы в составе некоторой фазы, как правило, делаются именно в этой фазе, однако вообще-то это не обязательно. Например, в ходе фазы управления можно писать новые истории. Кроме того, смена фаз осуществляется циклически: после того, как вы в течение некоторого времени находитесь в фазе управления, необходимо вновь вернуться к исследованию.

Фаза исследования

Фаза исследования предназначена для того, чтобы предоставить обеим сторонам возможность уяснить, что должна делать вся система. Фаза исследования включает в себя три хода.

- *Написание истории* — бизнес пишет историю, в которой описывается нечто, что должна делать система. Истории записываются на специальных карточках, где указывается имя и присутствует короткий абзац, описывающий цель истории.
- *Оценка истории* — разработчики делают оценку времени, которое потребуется для реализации истории. Если разработчики не могут оценить историю, они просят бизнес предоставить дополнительные разъяснения либо разделить историю. Чтобы оценить историю, можно спросить самого себя: «Какое время потребовалось бы мне для того, чтобы реализовать историю, если эта история была бы всем, что мне необходимо реализовать, и при этом меня никто не отвлекал бы и мне не надо было бы ходить на совещания?» В рамках XP мы обозначаем данный показатель термином «идеальное время разработки» (Ideal Engineering Time). Как будет показано позже (в разделе «Определение скорости»), прежде чем приступить к составлению графика работ, вы определяете коэффициент между идеальным временем и календарным.
- *Разделение истории* — если разработчики не могут оценить всю историю или если бизнес приходит к выводу, что некоторая часть истории является более важной, чем остальная история, бизнес может разделить историю на две или более историй.

Фаза подтверждения

В фазе подтверждения бизнес должен определить объем работ и дату выхода следующей версии, а разработчики должны со всей уверенностью подтвердить, что они в состоянии выполнить намеченный объем работ к заданному сроку. Фаза подтверждения включает в себя четыре хода.

- *Сортировка в соответствии сценностью* — бизнес разделяет истории на три категории: (1) истории, без которых система не сможет функционировать, (2) истории, которые являются менее важными,

но обеспечивают значительную полезность для бизнеса, (3) истории, которые было бы неплохо реализовать в рамках программного продукта.

- *Сортировка в соответствии с риском* — разработчики разделяют истории на три категории: (1) истории, которые можно оценить с высокой степенью точности, (2) истории, которые можно оценить с приемлемой точностью, (3) истории, которые невозможно оценить.
- *Определение скорости* — разработчики сообщают бизнесу, насколько быстро команда сможет программировать, оценка скорости выполняется отношением идеального времени разработки к календарному месяцу.
- *Определение объема работ* — бизнес выбирает набор карт для очередной версии. Для этого он либо устанавливает дату завершения работы над версией и выбирает карты в соответствии с их оценкой и скоростью работы над проектом, либо выбирает карты в соответствии с оценкой, а затем определяет дату завершения работы.

Фаза управления

Фаза управления предназначена для обновления плана на основе новых данных, которые появляются у разработчиков и у бизнеса. Фаза управления включает в себя четыре хода.

- *Итерация* — в начале каждой итерации (одна итерация длится в течение от одной до трех недель) бизнес выбирает несколько наиболее ценных для него историй, которые будут реализованы в рамках данной итерации. В результате реализации историй самой первой итерации должна получиться работающая от начала до конца система, пусть даже в самом зачаточном состоянии.
- *Регенерация* — если разработчики приходят к выводу, что они переоценили собственную скорость, они спрашивают у бизнеса, какой набор наиболее важных историй следует сохранить в рамках текущей версии. При определении этого набора учитывается новая скорость и оценки.
- *Новая история* — если в середине работы над очередной версией бизнес приходит к выводу, что в версию необходимо добавить новую историю, он может написать эту историю. Разработчики оценивают историю, после чего бизнес убирает из оставшегося плана истории с эквивалентной суммарной оценкой и добавляет в план новую историю.
- *Переоценка* — если разработчики приходят к выводу, что план больше не соответствует точной картине разработки, они могут заново

оценить оставшиеся истории и заново определить скорость разработки.

Итерационное планирование

Описанная в предыдущем разделе игра в планирование (Planning Game) предоставляет заказчику возможность управлять процессом разработки каждые три недели. В рамках одной итерации разработчики применяют фактически те же правила для того, чтобы планировать свои собственные действия.

Игра в итерационное планирование (Iteration Planning Game) очень напоминает игру в планирование (Planning Game), в которой карты используются в качестве кусков (pieces). На этот раз в качестве кусков используются не карты историй (story cards), а карты задач (task cards). Игроками являются отдельные программисты. Шкала времени короче — вся игра укладывается в одну итерацию (от одной до четырех недель). Фазы и ходы схожи с фазами и ходами игры в планирование.

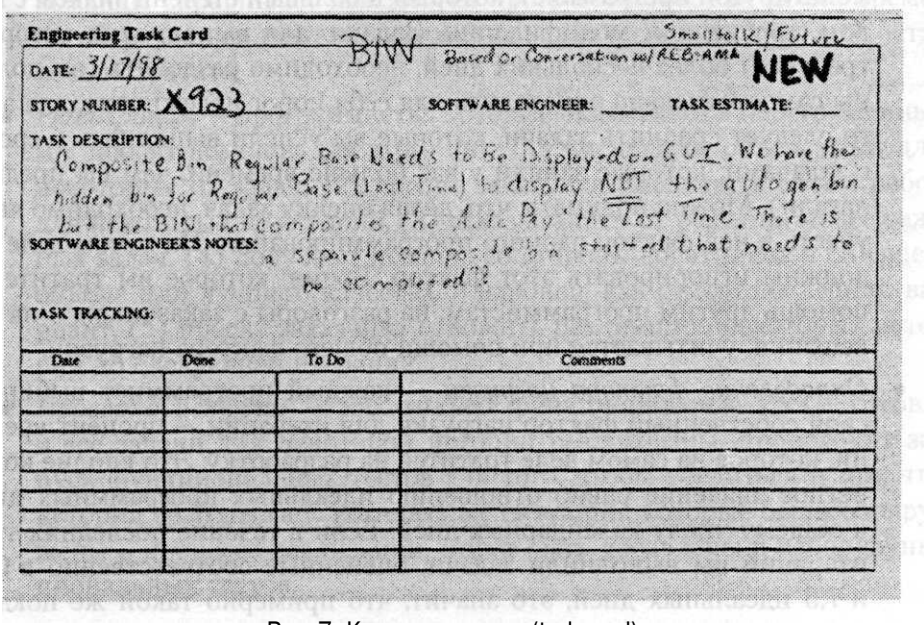


Рис. 7. Карточка задачи (task card)

Фаза исследования

- *Написание задачи* — выбор историй для итерации и преобразование их в задачи. Как правило, задача — это нечто меньшее по масштабу, чем история, так как нельзя реализовать одну историю целиком все-

го за пару дней. Иногда одна задача служит для реализации нескольких историй. Иногда задача не связана напрямую с какой-либо историей — например, переход на использование новой версии системного программного обеспечения. На рис. 7 показан пример реальной карточки задачи (task card).

- *Разделение задачи/комбинация задач* — если вы не можете оценить задачу в несколько дней, разделите ее на более мелкие задачи. Если выполнение нескольких задач потребует всего несколько часов, вы можете скомбинировать их в одну большую задачу.

Фаза подтверждения

- *Принятие задачи* — программист принимает на себя ответственность за выполнение задачи.
- *Оценка задачи* — ответственный программист оценивает количество идеальных дней разработки, необходимых для реализации каждой из задач. Часто эта оценка зависит от того, сможет ли оказать помощь другой программист, который в большей степени знаком с кодом, требующим модификации. Задачи, для выполнения которых требуется более нескольких дней, необходимо разделить (вы должны самостоятельно определить для себя порог надежности, для этого следует сравнить задачи, которые вы успели выполнить к сроку, с задачами, которые заняли у вас больше времени, чем вы предполагали). Можно подумать, что, делая оценку задач, необходимо явно учесть в ней фактор парного программирования. На самом деле вы должны игнорировать этот фактор. Время, которое вы тратите на помощь другим программистам, на разговоры с заказчиком и на совещания, учитывается при помощи общего фактора нагрузки.
- *Определение фактора нагрузки* — каждый программист выбирает свой собственный фактор нагрузки для итерации — процент времени, которое на самом деле тратится на разработку. Это вполне конкретное значение равно отношению идеальных программных дней к общему числу календарных дней. Если в течение последних трех итераций вы выполнили задачи, оцененные соответственно в 6, 8 и 7,5 идеальных дней, это значит, что примерно такой же показатель следует использовать и для очередной следующей итерации. Для новых членов команды, равно как и для инструктора XP, количество идеальных дней программирования в течение одной итерации может быть совсем небольшим — 2 или 3 дня в течение трехнедельной итерации. Для всех остальных членов команды это значение не должно быть больше, чем 7 или 8 дней. Если для какого-то про-

граммиста это значение больше, это означает, что он слишком мало времени тратит на помощь своим соратникам.

- *Балансировка* — каждый программист складывает оценки для своих задач и умножает на фактор нагрузки. Программисты, которые оказываются перегруженными, отказываются от части своих задач. Если перегруженной оказывается вся команда, необходимо найти способ сбалансировать ситуацию (см. подраздел «*Регенерация*» далее по тексту).

Фаза управления

- *Реализация задачи* — программист берет карточку задачи, находит партнера, пишет тестовые случаи для задачи, добивается их срабатывания, затем интегрирует новый код в систему, и когда весь универсальный набор тестов срабатывает, новый код делается доступным для остальных программистов.
- *Отслеживание прогресса* — каждые два или три дня один из членов команды спрашивает каждого из программистов, как много времени потрачено на решение каждой из задач и сколько дней еще осталось потратить.
- *Регенерация* — программисты, которые оказываются перегруженными, просят помощи; для этого они: (1) уменьшают объем работ для некоторых из задач, (2) просят заказчика уменьшить объем работ для некоторых из историй, (3) отказываются от решения менее важных задач, (4) получают помощь со стороны соратников в большем объеме или лучшего качества, и наконец, как самый последний вариант, (5) просят заказчика отложить реализацию некоторых историй до следующей итерации.
- *Проверка истории* — как только функциональные тесты готовы и все задачи для некоторой истории реализованы, происходит запуск функциональных тестов для того, чтобы убедиться в том, что история срабатывает. Интересные ситуации, которые были обнаружены в процессе реализации, могут быть добавлены в набор функциональных тестов.

Различие между планированием итерации и планированием версии состоит в том, что формирование плана итерации может быть более гибким. Если прошла одна из трех недель итерации и процесс идет слишком медленно, возможно, имеет смысл остановиться на один день и выполнить всеобщую совместную переработку кода для того, чтобы обеспечить дальнейший прогресс. Как правило, после нескольких таких ситуаций

у программистов не складывается впечатление, что весь проект разваливается на части. Однако если заказчики наблюдают подобные изменения, которые происходят с проектом изо дня в день, это заставляет их нервничать.

В некотором роде это выглядит как ложь, так как вы утаиваете некоторую часть процесса разработки от заказчика. Однако на самом деле это не так. Вы ничего не утаиваете преднамеренно. Если заказчик желает весь день сидеть рядом с вами и наблюдать, как осуществляется переработка кода системы, пожалуйста: пусть сидит, хотя возможно, у него есть более важные дела. Различие между планированием в рамках итерации и между итерациями — это расширение принципа разделения решений между бизнесом и разработчиками. На определенном уровне детализации существуют изменения, которые не должны беспокоить бизнес, — программисты отлично знают, как лучше осуществлять микроуправление своим рабочим временем, бизнесмен вряд ли сделает это лучше.

Еще одним отличием между игрой в планирование и игрой в планирование итерации состоит в том, что программист выбирает задачу *перед* тем, как сделать оценку. Команда в явной форме берет на себя ответственность за реализацию решений, поэтому оценки в этом случае должны делаться всей командой коллективно. Отдельные программисты берут на себя ответственность за реализацию задач, поэтому они должны оценивать эти задачи самостоятельно в индивидуальном порядке.

Еще одним отличием планирования итерации является то, что некоторые задачи не связаны напрямую с потребностями заказчика. Если кто-либо желает улучшить инструменты, используемые для интеграции системы, и связанный с этим объем работ достаточно значителен, тогда эта проблема становится отдельной задачей, которая включается в график работ и которой назначается приоритет наравне с другими задачами.

Давайте вернемся к ограничениям, связанным с процессом планирования итерации, и рассмотрим, как описанная ранее стратегия удовлетворяет этим ограничениям.

- Вы не желаете тратить слишком много времени на планирование, так как реальность никогда не соответствует плану с точностью 100%. Половина дня из пятнадцати — это не такая уж серьезная потеря. Конечно же, если вы можете сделать это время еще меньше, это будет лучше, однако половина дня — это действительно не так уж и много.
- Вы желаете обладать быстрой и надежной обратной связью относительно того, как идут дела, благодаря чему спустя треть от намеченного времени работы вы можете определиться, существуют ли у проекта проблемы. Ответы на вопросы, которые задает ревизор (tracker),

позволяют вам получить неплохое представление о том, отстаете ли вы от графика или нет. Как правило, оставшегося времени хватает на то, чтобы должным образом прореагировать на проблемы локально, то есть не обращаясь к заказчику с просьбой об изменениях.

- Вы желаете, чтобы люди, ответственные за разработку чего-либо, также выполняли предварительную оценку этой работы. Если программисты будут брать на себя ответственность за выполнение задач перед тем, как выполнить оценку этих задач, это требование будет удовлетворено.
- Вы желаете ограничить объем разработки теми частями проекта, которые действительно нужны. Это всегда звучит странно, когда вы говорите, что на протяжении трех недель вы можете работать только в течение 7,5 дней (15 дней разделить на измеренный фактор нагрузки, равный 2). Однако по мере того, как вы будете учиться формировать точные оценки, вы обнаружите, что это — абсолютная правда. Ощущение, что вы не работаете с максимально возможной интенсивностью, стимулирует у вас желание взять на себя выполнение большего количества задач. Однако при этом вы знаете, что вам надо поддерживать существующие стандарты и приемлемый уровень качества (и у вас есть партнер, который смотрит на тот же самый экран и следит за тем, чтобы вы работали качественно). Таким образом, вы получаете тенденцию работать просто и все также с гордостью могли бы сказать, что вы завершили работу.
- Вы желаете получить процесс, который не генерировал бы столь большого давления на людей. Так как люди под давлением начинают совершать глупые поступки лишь для того, чтобы достигнуть краткосрочной поставленной перед ними цели. При этом о долгосрочных целях они просто не задумываются. И снова это восходит к разговору о 7,5 идеальных рабочих днях за три календарные недели работы. Вы просто не можете взять на себя слишком большое количество задач. В результате вы берете на себя столько задач, сколько вы сможете сделать, обеспечив при этом приемлемый уровень качества.

Для небольших проектов я не использую итерационного планирования. Очевидно, что планирование итераций необходимо для координации работы команды из десяти программистов. Итерационное планирование не требуется в случае, если в проекте задействовано всего два программиста. В зависимости от масштаба проекта вы поймете, что необходимость координации оправдывает дополнительные усилия, затрачиваемые на итерационное планирование.

Планирование за неделю

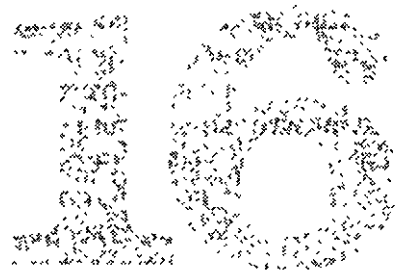
Как планировать проект, если у вас в запасе всего неделя? Подобная ситуация часто возникает в случае, если команда должна сделать предложение с фиксированной ценой. Вы получаете тендер, и у вас есть всего неделя, чтобы среагировать. У вас нет времени для того, чтобы написать полный набор историй, каждую из которых вы могли бы оценить и протестировать. У вас нет времени на написание прототипов, поэтому вы должны оценить истории исходя из личного опыта.

Решение в рамках XP предусматривает принятие большего риска в плане при помощи более крупных историй. Напишите истории, которые можно оценить в терминах не недель, а месяцев идеального программирования. Вы можете предоставить заказчику возможность выбирать, для чего он может либо сузить, либо отложить некоторые из историй (как и в обычной игре в планирование).

Оценки необходимо делать исходя из опыта. Если вам надо ответить в течение недели, то в отличие от обычной игры в планирование вы не обладаете временем, достаточным для того, чтобы сформировать опыт. Вы должны оценивать исходя из предыдущего опыта разработки подобных систем. Если у вас нет предыдущего опыта разработки аналогичных систем, значит, вы не можете заниматься этим бизнесом.

После того как вы получите контракт, первая вещь, которую вам необходимо сделать, это вернуться обратно к начальным стадиям игры в планирование, благодаря чему вы немедленно проверите свою возможность выполнить условия контракта в срок.

Стратегия разработки



В отличие от стратегии менеджмента стратегия разработки радикально отличается от того, что принято считать общепризнанной мудростью, — мы будем тщательно формировать решение сегодняшней проблемы именно сегодня в надежде на то, что мы всегда сможем решить завтрашнюю проблему завтра.

В XP метафора программирования используется для всех видов деятельности, благодаря чему все, чем вы занимаетесь, выглядит как программирование: программирование в XP выглядит как программирование с некоторыми небольшими добавлениями, например, автоматическим тестированием. Однако разработка, как и все остальное в XP, может показаться простой только на первый взгляд. Все составные части достаточно просты, чтобы их объяснить, однако использовать их совместно достаточно сложно. Как только вы начинаете практиковать XP, у вас появляется страх. Как только появляется давление, возвращаются старые привычки.

Стратегия разработки начинается с планирования итерации. Постоянно выполняемая интеграция уменьшает количество конфликтов и является естественным завершением эпизода разработки. Коллективное владение стимулирует всю команду делать всю систему все лучше и лучше. Наконец, программирование парами связывает весь процесс в единое целое.

Постоянная интеграция

Никакой код не остается не интегрированным в систему в течение более чем двух часов. В завершение каждого эпизода разработки полученный код интегрируется в текущую версию системы, и при этом все тесты должны сработать на все 100%.

При идеальной постоянно продолжающейся интеграции каждый раз, когда вы изменяете метод, это изменение должно немедленно отражаться во всем остальном, пусть даже не принадлежащем вам коде. Если даже не принимать во внимание инфраструктуру и скорость обмена информацией, которые необходимы для обеспечения работы в таком стиле, это все равно может не сработать. Когда вы разрабатываете код, вы желаете чувствовать себя единственным программистом, работающим над проектом. Вы хотите нестись вперед на полной скорости, игнорируя взаимосвязи между изменениями, которые делаете вы, и изменениями, которые делают все остальные участники проекта. Когда изменения выходят из-под вашего контроля, иллюзия разрушается.

Интеграция через каждые несколько часов (не чаще чем через день) предоставляет множество преимуществ для обоих стилей — единственный программист и немедленная интеграция. Когда вы разрабатываете код, вы можете действовать так, как будто вы и ваш партнер — это единственная пара, работающая над проектом. Вы можете вносить в проект изменения тогда, когда вы этого хотите. После этого роли меняются. Когда вы приступаете к интеграции, вы узнаете (специальные средства сообщают вам об этом) о том, в каких именно определениях классов и методов возникли конфликты. Запустив тесты, вы узнаете о семантических конфликтах.

Если интеграция отнимает у вас пару часов, работать в таком стиле становится невозможно. Чтобы обеспечить эффективную работу, необходимо использовать инструменты, которые обеспечивают быстрый цикл интеграция/сборка/тестирование. Вы также должны обладать достаточно быстрым набором тестов, для срабатывания которого требуется всего несколько минут. Усилия, которые вам потребуются для того, чтобы устранить конфликты, не должны быть слишком большими.

На самом деле это не такая уж серьезная проблема. В результате постоянной переработки кода система разбивается на множество небольших объектов и множество небольших методов. Благодаря этому снижается вероятность того, что две пары программистов в одно и то же время занимаются модификацией одного и того же метода или класса. Если же это на самом деле происходит, чтобы уладить возникшие при этом конфликты, требуется приложить совсем небольшие усилия, потому что каждая из версий одного и того же кода была создана в течение всего нескольких часов разработки.

Еще одной важной причиной, по которой следует смириться с затратами, вызванными постоянной интеграцией, является то, что при этом существенно снижается общий риск всего проекта. Если у двух разных людей в голове рождаются две разные идеи о том, как должен выглядеть

или функционировать некоторый кусок кода, вы узнаете об этом буквально через несколько часов. То есть вам не придется тратить несколько дней иа поиск ошибки, которая возникла уже достаточно значительное время назад — в ходе двух-трех минувших недель работы над программой. Кроме того, вся эта практика постоянной интеграции оказывается чрезвычайно полезной при создании финальной рабочей версии проекта. Оказывается, что сборка готовой к поставке заказчику версии — это не такая уж серьезная проблема. Любой программист в команде сможет собрать финальную версию почти что с завязанными глазами, так как все они занимались этим по несколько раз на дню в течение нескольких месяцев.

Постоянно продолжающаяся интеграция обеспечивает значительное преимущество также с точки зрения человеческого фактора. В процессе работы над задачей в вашей голове рождается множество связанных с этим мыслей. Когда вы завершаете решение задачи и приступаете к интеграции, это означает, что ваш список дел, которые вы должны сделать, очистился. Вы очищаете свою голову от вещей, которые предстоит сделать, и приступаете к анализу того, что получилось в результате. Таким образом, интеграция обеспечивает естественный ритм разработки. Размышление/тестирование/кодирование/выпуск готового кода. Это почти так же естественно, как дыхание. Вы формируете идею, вы формулируете ее, затем вы добавляете ее в систему. Теперь ваша память свободна и готова к следующей идее.

Время от времени постоянно выполняемая интеграция принуждает вас разделить реализацию задачи на два эпизода. Необходимо принять дополнительные связанные с этим затраты и помнить, что уже сделано, а что только предстоит сделать. В промежутке времени между двумя эпизодами возможно у вас появится предположение о том, почему первый эпизод затянулся столь надолго. Вы можете начать следующий эпизод с переработки кода, и тогда весь второй эпизод пройдет более гладко.

Коллективное владение

Коллективное владение — это на первый взгляд бредовая идея, предполагающая, что кто угодно имеет право изменять любой кусок кода в каком угодно месте системы в любое удобное для этого время. Если вы не практикуете автоматическое тестирование, применять коллективное владение на практике — это верный способ самоубийства. Однако с использованием тестирования и благодаря тому, что качество ваших тестов спустя месяцы практики в этом направлении будет достаточно высоким, вы може-

те смело использовать коллективное владение. Для того чтобы без опаски применять коллективное владение, необходимо также осуществлять интеграцию каждые несколько часов работы над проектом. Именно всем этим, конечно же, вы и будете заниматься в рамках XP.

Одним из эффектов коллективного тестирования является то обстоятельство, что сложный код не живет в системе слишком долго. Все программисты команды постоянно просматривают систему, рано или поздно они обнаруживают сложный код, и когда это происходит, обязательно находится кто-то, кто попытается упростить этот сложный код. Если новая, упрощенная версия кода не срабатывает (что демонстрируется несрабатыванием тестов), новый код отбрасывается в сторону. Даже если подобное происходит, это означает, что есть еще кто-то помимо изначально разрабатывавшей код пары, кто теперь понимает, почему этот код должен быть сложным. Однако чаще всего упрощение кода срабатывает полностью или по крайней мере частично.

Коллективное владение в первую очередь препятствует проникновению сложного кода в систему. Если вы знаете, что кто-то еще кроме вас в самом ближайшем времени (буквально через несколько часов) будет просматривать код, который вы сейчас пишете, вы дважды подумаете, прежде чем добавите в систему сложный код, которому вы не можете довериться прямо сейчас.

Коллективное владение усиливает ощущение вашей личной власти над проектом. В рамках XP-проекта вы никогда не проклинаете в бессилии чужую тупость, мало того, чужая тупость никогда не становится непреодолимой преградой на вашем пути. Если вы видите на своем пути какое-либо препятствие, вы просто избавляетесь от него. Если вы предпочитаете сохранить что-либо, потому что это вам подходит, — это ваше личное дело. Но при этом вы никогда не попадаете в тупик. У вас никогда не возникает ощущения, что: «Я мог бы отлично справится с моей работой, если бы только не все эти идиоты вокруг меня». А это значит, что у вас исчезает еще одна причина для огорчения. Вы еще на один шаг ближе к чистому мышлению.

Коллективное владение также способствует распространению знаний о системе среди членов команды. Очень маловероятно, что в системе найдется какая-либо часть, о строении которой будут знать только два человека (это должна быть по крайней мере пара, что уже лучше, чем распространенная ситуация, когда один очень умный программист держит всех остальных в заложниках). А это еще в большей степени снижает риск проекта.

Программирование парами

Программирование парами действительно заслуживает того, чтобы о нем написали отдельную книгу. Это изящное искусство, на совершенствование которого вы можете потратить всю оставшуюся жизнь. В данной главе я лишь расскажу вам о том, почему программирование парами используется в рамках XP.

• Для начала скажу, чем парное программирование *не* является. Парное программирование — это не значит, что один человек программирует, а другой смотрит. Смотреть на то, как кто-то программирует, и ничего при этом не делать — это также интересно, как смотреть на то, как зеленая трава умирает посреди иссушенной пустыни. Программирование в паре — это диалог между двумя людьми, пытающимися разрабатывать одновременно один и тот же код (и при этом анализировать, проектировать и тестировать), а также возможность совместно понять, как этот код можно запрограммировать лучше. Программирование в паре — это обмен информацией на нескольких уровнях, осуществляемый при помощи компьютера и сфокусированный на компьютере.

Также следует отметить, что программирование в паре — это не сеанс обучения. Иногда в состав пары входят два партнера, один из которых обладает существенно большим опытом, чем другой. Когда такое происходит, первые несколько сеансов парного программирования во многом напоминают уроки. Менее опытный партнер задает множество вопросов и вводит в компьютер относительно немного кода. Однако через короткое время менее опытный партнер начинает отлавливать грубые ошибки, такие как несоответствие открывающихся и закрывающихся скобок. Более опытный партнер заметит, что оказываемая им помощь востребована. Еще через несколько недель менее опытный партнер приступит к использованию более крупных образцов кода, чем использует его более опытный соратник. Для него станет понятным разница между образцами кода.

Как правило, через пару месяцев пробел между двумя партнерами уже не столь заметен, как это было в самом начале. Менее опытный партнер регулярно занимается вводом нового кода. Члены пары замечают, что у каждого из них есть свои сильные и слабые стороны. Производительность, качество и удовлетворение растут.

Программирование в паре — это не объединение двух людей, которым скучно в одиночестве. Если вернуться к главе 2, можно вспомнить, что в самом начале я обращаюсь к своему соратнику за помощью. Иногда для того, чтобы решить определенную задачу, вам нужен определенный партнер. Однако чаще вы выбираете себе партнера из тех, кто доступен. И если у вас обоих есть задачи, которые необходимо выполнить, вы можете

согласиться с утра работать над одной задачей, а после полудня работать над другой задачей.

Что, если два человека не могут ужиться вместе? В этом случае они не должны входить в состав одной пары. Если в команде есть такие два человека, это может усложнить формирование пар. Если персональные проблемы в команде слишком значительны, лучше потратить несколько минут на корректное формирование пар, чем довести ситуацию до кулачного боя.

Что если кто-то отказывается работать в парах? Такие люди либо должны научиться работать так, как работает вся остальная команда, либо должны работать вне команды. XP подходит далеко не каждому, и далеко не каждый подходит для XP. Однако вовсе необязательно с самого первого дня внедрения XP все свое рабочее время программировать в паре и только в паре. Как и ко всему остальному в XP, вы должны привыкать к программированию в парах постепенно. Попробуйте работать в паре в течение часа. Если у вас плохо получится, попытайтесь понять, что именно вы делаете не так, затем попробуйте поработать в паре еще один час.

Почему программирование парами хорошо работает в рамках XP? Прежде всего потому, что программирование парами обеспечивает отличную коммуникацию. Мало того, программирование в парах обеспечивает более интенсивную коммуникацию, чем обмен сведениями между двумя людьми. Таким образом, программирование в парах хорошо работает в XP потому, что оно стимулирует коммуникацию. Я часто сравниваю команду с миской, в которую налита прозрачная вода. Как только кто-либо в команде узнает какую-либо важную новость, это напоминает падение капли краски в миску с прозрачной водой. Партнеры в парах все время меняются, поэтому важная свежая информация быстро распространяется среди членов команды подобно тому, как краска растворяется в воде. Однако в отличие от краски по мере распространения информация становится более насыщенной и более продуманной. Она насыщается за счет опыта и интеллекта каждого из членов команды.

Мой опыт показывает, что программирование в парах более продуктивно, чем разделение работы между двумя отдельными программистами и последующая интеграция результатов. Программирование парами часто становится на первый план для людей, желающих использовать XP. Все, что я могу посоветовать, это то, что вы должны вначале хорошо овладеть им, затем попробовать выполнить очередную итерацию продукта так, чтобы весь код был написан в парах, а затем следующую итерацию разработать так, чтобы программисты работали по отдельности. После этого вы сможете сформировать ваше собственное мнение.

Даже если вы не получили роста продуктивности, вы все равно не должны сбрасывать со счетов программирование в парах, потому что получаемый в результате этого код обладает существенно более высоким качеством. Пока один из партнеров занят набиванием кода на клавиатуре, другой партнер размышляет на более высоком стратегическом уровне. Куда ведет избранный путь кодирования? Не попадем ли мы в тупик? Существует ли более эффективная стратегия организации этого кода? Какие есть возможности для переработки кода?

Еще одной важной особенностью программирования парами является то обстоятельство, что некоторые из методик XP без него не работают. В состоянии стресса люди забывают о правилах хорошего тона. Они перестают писать тесты. Они отказываются от выполнения переработки. Они откладывают на потом интеграцию. Однако если за вами следит ваш партнер и при этом вы собираетесь нарушить одну из методик, у вас возникает чувство вины. Вам неудобно перед вашим партнером, так как вам кажется, что он никогда бы так не поступил. Сказанное не означает, что пары никогда не нарушают нормальный ход процесса разработки. Конечно же иногда это происходит, в противном случае вам не понадобился бы инструктор (coach). Однако если вы работаете в паре, вероятность того, что вы проигнорируете некоторые общепринятые правила, ниже, чем если бы вы работали в одиночку.

Противоречивая природа программирования парами также существенно улучшает процесс разработки программного обеспечения. Вы быстро учитесь разговаривать на нескольких разных уровнях — этот код здесь, код, подобный этому, в другом месте системы, эпизоды разработки, подобные этому, в прошлом, системы, подобные этой, над которыми вы работали несколько лет назад, методики, которые вы используете, и как их можно улучшить.

Стратегия проектирования 17

Мы начнем с самого простого **дизайна,** который только возможен. После этого мы будем постоянно пересматривать дизайн системы. Мы будем удалять из системы любую гибкость, которая оказывается бесполезной.

Во многих отношениях эта глава оказалась для меня самой сложной из всей книги. Стратегия дизайна в XP предусматривает, что система всегда должна обладать наиболее простым дизайном, при которым срабатывает текущий набор тестов.

Рассмотрим подробнее, что такое простота и что такое наборы тестов.

Самая простая вещь, которая, возможно, сработает

Давайте сделаем шаг назад и подойдем к решению проблемы постепенно. В формировании этой стратегии участвуют все четыре ценности.

- *Коммуникация* — сложный дизайн сложнее описать, чем простой. По этой причине мы должны создать стратегию проектирования, которая формирует наиболее простой возможный дизайн, согласующийся со стоящими перед нами целями. С другой стороны, мы должны создать стратегию дизайна, которая формирует описательные и информативные дизайны, элементы которого хорошо описывают внутреннее строение системы тому, кто изучает этот дизайн.
- *Простота* — мы собираемся сформировать стратегию, которая помогала бы нам создавать простые дизайны, однако при этом, и сама стратегия должна быть простой. Это не означает, что она должна просто воплощаться в жизнь. Хороший дизайн — это всегда не так уж просто. Однако объяснить стратегию должно быть просто.

- *Обратная связь* — одна из проблем, с которой мне приходилось сталкиваться в процессе проектирования, прежде чем я начал практиковать XP, состояла в том, что я никогда не мог сказать точно, прав я или нет. Чем дольше я проектировал, тем значительней становилась эта проблема. Простой дизайн решает проблему благодаря тому, что он формируется быстро. Далее следует закодировать его и посмотреть, как выглядит и ведет себя код.
- *Храбрость* — что может быть более отважным, чем остановиться, обладая лишь частью дизайна, приступить к реализации и быть уверенным в том, что в дальнейшем вы сможете добавить в систему больше, если в этом возникнет необходимость.

Следуя этим ценностям, мы должны:

- сформировать стратегию проектирования, в результате использования которой формируется простой дизайн;
- найти быстрый способ убедиться в том, что дизайн качественный;
- организовать обратную связь для того, чтобы быстро воплощать наши новые открытия и выводы в дизайне системы;
- сжать цикл времени, в течение которого выполняется весь этот процесс, и сделать его как можно короче.

Принятые нами ранее принципы также хорошо воплощаются в стратегии дизайна.

- *Небольшие изначальные инвестиции* — прежде чем получить первую отдачу от дизайна, мы должны инвестировать в проектирование системы так мало, насколько это возможно.
- *Приемлемая простота* — мы должны быть уверенными в предположении, что самый простой дизайн, решающий проблему, который мы только можем себе представить, скорее всего, будет работать. Благодаря этому мы получим дополнительное время на решение возникших проблем в случае, если сформированный нами простой дизайн не срабатывает. Кроме того, благодаря такому подходу нам не придется тратить дополнительные ресурсы на обеспечение дополнительной гибкости.
- *Постепенное изменение* — стратегия дизайна будет работать благодаря постепенному изменению. Мы будем проектировать постепенно и понемногу. Никогда не наступит момент времени, когда можно будет сказать, что «система полностью спроектирована». Дизайн системы будет постоянно меняться, однако при этом некоторые части системы, возможно, будут оставаться неизменными в течение некоторого времени.

- *Путешествие налегке* — стратегия проектирования не должна формировать какого-либо «лишнего» дизайна. Дизайн должен быть достаточным для того, чтобы решать наши текущие задачи (необходимость делать качественную работу), но не более того. Если нам придется постоянно все менять, мы должны иметь возможность начать с самого простого и постоянно пересматривать то, что у нас имеется на текущий момент.

XP работает против многих программистских инстинктов. Мы, программисты, привыкли ожидать появления проблем. Если проблемы откладываются на более позднее время, мы счастливы. Если проблемы не появляются, мы не обращаем на это внимания. Поэтому наша стратегия проектирования должна увести нас в сторону от этих «размышлений о будущем». К счастью, большинство разработчиков способно отучится от этой привычки «брать неприятности взаймы» (как про это говорила моя бабушка). К сожалению, чем вы умнее, тем сложнее вам отучиться от этого.

Еще один способ взглянуть на это предлагает заданный себе вопрос: «Когда следует добавить еще дизайна?» Общепринято отвечать на него, что вы должны думать о том, какие проблемы встанут перед вами завтра, и исходя из этого вы должны проектировать программу с расчетом на завтра (рис. 8).

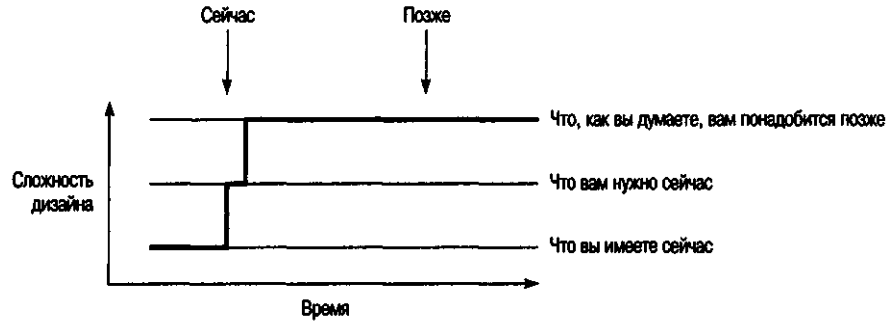


Рис. 8. Если стоимость затрат стремительно растет с течением времени

Эта стратегия работает в случае, если между сегодня и завтра ничего не меняется. Если вы точно знаете, что будет завтра, и вы точно знаете, как с этим справиться в большинстве случаев, сегодня вы должны добавить в систему то, что вам нужно сейчас, а также то, что вам потребуется завтра.

Проблема, связанная с этой стратегией, — это неопределенность. На практике:

- иногда «завтра не наступает никогда» (то есть возможность, которую вы спроектировали заранее, больше не интересует заказчика);

- иногда вы придумываете лучший способ перехода «от сегодня к завтра».

В любом случае, вы должны выбрать между затратами, необходимыми для того, чтобы убрать из системы ненужный дизайн, и затратами, необходимыми для того, чтобы продолжать разработку, имея на руках сложный дизайн, который не приносит вам пользы.

Я ничего не имею против изменений, вносимых заказчиком в план работ. Я также ничего не имею против того, чтобы со временем менять реализацию той или иной части системы в лучшую сторону. В этом случае картинку, изображенную на рис. 8, следует изменить. Мы должны проектировать систему так, чтобы сегодня решать те проблемы, которые стоят перед нами сегодня, и откладывать на завтра решение тех проблем, которые будут стоять перед нами завтра (рис. 9).

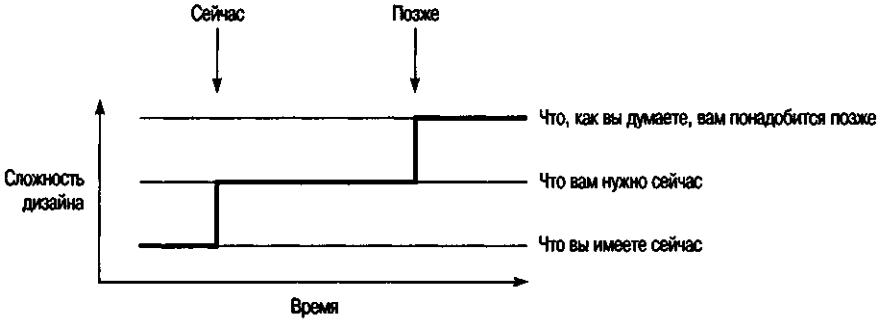


Рис. 9. Если с течением времени стоимость изменений остается невысокой

Это ведет нас к созданию следующей стратегии проектирования.

1. Вначале разрабатывается тест, благодаря чему у нас появляется возможность определить момент завершения работы. Для того чтобы просто написать тест, мы опять же должны выполнить некоторый объем проектирования: мы должны определить набор объектов, с которыми мы работаем, а также набор видимых методов для этих объектов.
2. Мы проектируем и реализуем только для того, чтобы обеспечить срабатывание тестов. Все только что разработанные нами тесты, а также все тесты, которые были разработаны ранее, должны сработать — это единственная цель, которую мы преследуем в процессе проектирования.
3. Повторяем.
4. Если мы видим возможность упростить наш дизайн, мы немедленно делаем это. Основные принципы, позволяющие нам определить

степень простоты дизайна, рассматриваются в разделе «Что является самым простым?»

Эта стратегия может показаться смехотворно простой. И действительно, она очень проста. Но она не смехотворна. Используя данную стратегию, вы можете создавать большие сложные системы. Однако это непросто. Ничего нет сложнее, чем работать в рамках строгих ограничений по времени и при этом всегда находить время «чистить» код.

Проектируя в данном стиле, при решении некоторой задачи вы реализуете необходимый код самым простым возможным способом. Когда вы используете этот код повторно, вы делаете его более универсальным. При первом использовании код делает только то, что требуется. При повторном использовании код делается более гибким. При таком подходе вы никогда не платите за гибкость, которую вы не используете, кроме того, система имеет тенденцию становиться гибкой тогда, когда она должна становиться гибкой для третьей, четвертой и пятой вариаций.

Как работает «проектирование при помощи переработки»?

Если попробовать реализовать эту стратегию на практике, поначалу она покажется вам странной. Мы берем первый тестовый случай. Мы говорим: «Если нам надо только лишь обеспечить срабатывание этого теста, тогда нам потребуется всего один объект с двумя методами». Мы создаем объект, добавляем в него два необходимых метода и считаем дело сделанным: весь наш дизайн — это один объект. Но только на минуту.

После этого мы берем второй тестовый случай. Мы можем попытаться решить задачу, используя то, что есть у нас на руках, однако вместо этого, возможно, будет удобнее преобразовать существующий объект, разбив его на два разных объекта. В этом случае для обеспечения срабатывания тестового случая необходимо заменить один из полученных объектов. Поэтому прежде, чем продолжать работу, мы выполняем реструктуризацию нашей программы, затем мы проверяем, срабатывает ли наш первый тестовый случай, затем мы добиваемся срабатывания второго тестового случая.

После пары дней работы в таком режиме система становится достаточно большой, и мы уже можем представить себе две группы разработчиков, которые могут работать над ней, не наступая при этом постоянно друг другу на пятки. И тогда мы пускаем в дело две пары программистов, которые занимаются реализацией тестовых случаев параллельно друг с другом и периодически (через каждые несколько часов) интегрируют

вносимые ими изменения. Еще один или два дня, и система разрастается настолько, что мы можем обеспечить работой всю команду. Постепенно все члены команды начинают работать в описанном стиле.

Время от времени у команды будет возникать ощущение, что перед ними простирается невозделанная целина. Возможно, они обнаруживают существенное отклонение реальности от предварительных оценок. А может быть, их желудки завязываются узлом каждый раз, когда они приходят к выводу, что некоторая часть системы требует полной переработки. В любом случае, кто-то просит тайм-аут. Команда собирается вместе на целый день и плотно занимается реструктуризацией системы как единого целого, используя при этом комбинацию карт CRC, набросков и переработки кода.

Не каждую переработку можно выполнить за пару минут. Если вы обнаружили, что построили запутанную иерархию наследования классов, возможно, вам потребуется месяц на то, чтобы распутать ее. Однако у вас в запасе нет лишнего месяца. Вы обязаны реализовать все истории, запланированные для данной текущей итерации.

Когда вы сталкиваетесь с необходимостью крупномасштабной переработки кода, вы должны действовать небольшими шажками (вновь постепенное изменение). Вы работаете над некоторым тестовым случаем и видите возможность на один маленький шажок приблизиться к вашей большой крупномасштабной цели. Сделайте этот маленький шажок. Переместите метод сюда, переменную — туда. Постепенно крупномасштабное изменение станет не таким уж и крупномасштабным. После постепенной поэтапной эволюции вы сможете завершить переработку за пару минут.

В свое время я столкнулся с необходимостью крупномасштабной переработки кода, когда работал над системой управления страховыми контрактами. Тогда я попробовал выполнить ее небольшими шажками. У нас была иерархия, показанная на рис. 10.

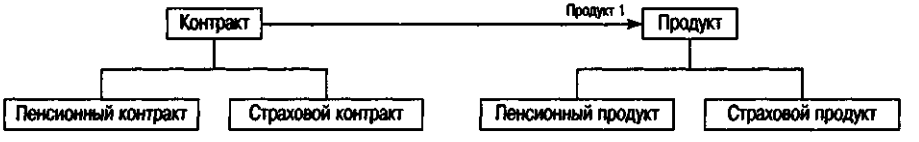


Рис. 10. Контракт и продукт обладают параллельными подклассами

Этот дизайн нарушает правило Once And Only Once (OAOO) объектно-ориентированного дизайна (код должен присутствовать в системе один раз и только один раз). Чтобы исправить ситуацию, мы начали работать над формированием дизайна, показанного на рис. 11.

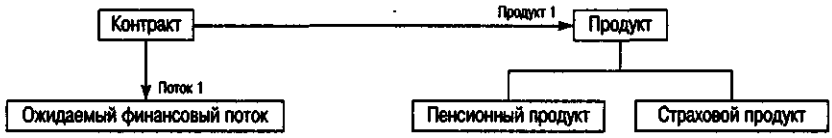


Рис. 11. Контракт ссылается на класс Function (функция), но не имеет подклассов

В течение года, пока мы работали над этой системой, мы сделали множество небольших шажков в направлении желаемого дизайна. Мы перекладывали обязанности подклассов класса Contract (контракт) либо на подклассы Function (функция), либо на подклассы Product (продукт). В самом конце работы над заказом мы не смогли полностью избавиться от подклассов Contract, однако они стали существенно менее содержательными, чем в начале, и было очевидно, что мы держим курс на отказ от их использования. Все это время мы продолжали добавлять в систему новую функциональность.

Вот так. Именно так осуществляется экстремальный дизайн. В рамках XP проектирование — это не рисование огромного количества схем и затем реализация системы в точном соответствии с этими схемами. В XP проектирование напоминает ориентирование автомобиля в нужном направлении в то время, как вы едете по шоссе. История об управлении автомобилем подсказывает нам совершенно иной стиль проектирования — вы заводите машину, начинаете движение, а затем поправляете руль чуть-чуть влево, затем чуть-чуть вправо, затем опять обратно влево.

Что является самым простым?

Таким образом, лучшим является самый простой дизайн, который обеспечивает срабатывание всех тестовых случаев. Чтобы сделать это определение эффективным, необходимо объяснить, что именно мы подразумеваем, когда говорим «самый простой».

Может быть, самый простой дизайн — это дизайн с наименьшим количеством классов? Но если в системе мало классов, значит используемые объекты становятся настолько большими, что их неудобно использовать. Может быть, самый простой дизайн — это дизайн с наименьшим количеством методов? Но это приведет к формированию слишком крупных методов, а следовательно — к дублированию кода. Может быть, самый простой дизайн — это дизайн с наименьшим количеством строк кода? Но тогда мы будем стремиться сжать программу только ради сжатия и, кроме того, нам потребуется слишком много общаться между собой.

Когда я говорю «самый простой дизайн», я имею ввиду следующие четыре ограничения в порядке приоритета.

1. Система (как ее код, так и соответствующие тесты) должна выражать собой все, что вы хотите сообщить о ней всем остальным участникам проекта.
2. Система не должна содержать дублирующегося кода (1 и 2 пункты вместе составляют собой правило Once and Only Once).
3. Система должна состоять из наименьшего возможного количества классов.
4. Система должна содержать в себе наименьшее возможное количество методов.

Цель проектирования системы — это, во-первых, выразить намерения программистов и, во-вторых, обеспечить место для размещения логики работы системы. Представленные здесь ограничения обеспечивают обрамление, в рамках которого необходимо удовлетворить двум этим условиям.

Если вы смотрите на дизайн как на среду обмена информацией, значит, вы должны создать объекты или методы для каждой важной используемой вами концепции. Вы должны выбрать имена классов и методов так, чтобы их было удобно использовать совместно.

Разрабатывая код, ограничивайте себя так, чтобы создаваемый вами код было бы удобно использовать для коммуникации, после этого вы должны удалить из системы весь дублирующийся код. Для меня это самая сложная часть проектирования. Дело в том, что вначале надо обнаружить дублирующийся код, а затем найти способ избавиться от дублирования. Для того чтобы избавиться от дублирования, как правило, приходится создавать множество мелких объектов и множество мелких методов, потому что в противном случае неизбежно возникнет дублирование кода.

Однако вы создаете новые объекты и новые методы не просто для собственного удовольствия. Если вы обнаруживаете класс, который ничего не делает и ни о чем не информирует, или метод, который ничего не делает и ни о чем не информирует, вы должны уничтожить их.

Еще один способ взглянуть на этот процесс — это провести аналогию со стиранием. У вас есть система, для которой срабатывают все тестовые случаи. Вы удаляете из нее все, что не имеет определенной цели — либо коммуникационной цели, либо вычислительной цели. То, что остается, — это самый простой дизайн, который, скорее всего, сработает.

Как это может работать?

Традиционная стратегия сокращения с течением времени затрат на разработку программного обеспечения заключается в том, чтобы снизить веро-

ятность переработки и затраты, связанные с переработкой. XP предлагает действовать с точностью до наоборот. Вместо того чтобы снижать частоту переработки, XP наслаждается переработкой. День без переработки — это день без солнечного света. Но как это может обходиться дешевле?

Ответ состоит в том, что риск — это деньги точно так же, как и время — это деньги. Если сегодня вы включаете в проект некоторую возможность и используете ее завтра, вы выигрываете, так как вы платите меньше за то, что включили эту возможность именно сегодня, а не завтра. Однако в главе 3, посвященной экономике разработки программного обеспечения, было показано, что эта оценка не является полной. Если сопутствующая этому неопределенность достаточно велика, ценность сценария, в котором вы просто ждете, может оказаться настолько большой, что вам становится выгодно просто подождать.

Дизайн не обходится вам бесплатно. Существует еще один важный аспект. Если сегодня вы формируете систему на основе более сложного дизайна, вы увеличиваете расходы, связанные с ее обслуживанием и поддержкой. Более сложный дизайн требует большего тестирования, больших усилий для понимания и объяснения. Поэтому каждый день вы платите не только процентную ставку, начисляемую на потраченные вами деньги, вы также выплачиваете небольшой налог на дизайн. При учете этого разница между сегодняшними инвестициями и завтрашними инвестициями становится еще более ощутимой. Таким образом, идея отложить решение завтрашних проблем до завтра выглядит более привлекательной.

Если вам не достаточно всех рассмотренных аргументов, я упомяну еще один — риск. Как было показано в главе 3, вы не можете точно оценить стоимость чего-либо, что произойдет завтра. Помимо связанных с этим затрат, вы должны оценить также вероятность того, что это действительно произойдет. Как и любой другой человек, я люблю делать предположения и оказываться правым, однако когда я стал внимательней следить за этим, я обнаружил, что я оказываюсь не прав приблизительно столь же часто, сколь часто я пытаюсь делать предположения. Зачастую сложный дизайн, который я разработал год назад, фактически не содержит в себе ни одного корректного предположения. Прежде чем я завершаю работу, я вынужден переделывать каждую часть моего проекта, иногда по два или три раза.

Затраты, связанные с решением, которое мы формируем сегодня, включают в себя стоимость решения плюс процентная ставка на сумму, которую мы тратим на реализацию этого решения, плюс стоимость инерции, которая добавляется в систему в результате воплощения этого решения. Преимуществом того, что мы воплощаем решение именно сегодня, явля-

ется ожидаемая ценность этого решения, которое мы, возможно, сможем с выгодой использовать завтра.

Если стоимость сегодняшнего решения высока, вероятность того, что оно окажется правильным, низка, вероятность того, что завтра вы найдете лучший способ решить проблему, высока, а стоимость внесения изменений в дизайн завтра низка, то мы можем прийти к выводу, что если сегодня мы можем обойтись без решения, значит, мы ни в коем случае не должны принимать это решение сегодня. Именно такой подход используется в рамках XP. «Количество сложностей ровно на один день и не более того».

Однако некоторые факторы могут стереть наши выводы в порошок. Если затраты, которые возникнут в случае, если мы будем принимать решение завтра, существенно больше сегодняшних, значит, мы должны принять решение сегодня в надежде на то, что завтра мы окажемся правы. Если инерция дизайна достаточно низка (над проектом работают очень-очень умные люди), значит, у дизайна, формируемого по мере разработки, остается все меньше и меньше преимуществ. Если вы действительно очень хороший провидец, значит, вы можете спроектировать все без исключения с самого начала, а затем приступать к реализации готового завершенного плана. Однако для всех остальных обычных людей я не вижу иной альтернативы, кроме той, в рамках которой предлагается проектировать сегодня только то, что требует проектирования именно сегодня, и откладывать на завтра то, что можно спроектировать завтра.

Роль рисунков в дизайне

Что можно сказать о графическом представлении дизайна, а также о визуальном проектировании и анализе? Некоторым людям действительно удобнее размышлять о структуре системы при помощи визуальных образов, а не строк кода. Как может визуально-ориентированный человек осуществлять проектирование системы?

Прежде всего хочу отметить, что если вместо чисто ментального или текстового представления вы проектируете систему с использованием графических изображений, в этом нет абсолютно ничего плохого. О визуальном подходе к проектированию следует сказать особо. Проблемы, которые возникают при рисовании графических диаграмм, могут служить подсказками, указывающими вам на состояние здоровья вашего дизайна. Если вы не можете найти способ уменьшить количество графических элементов на диаграмме, если существует явная асимметрия, если количество линий значительно превышает количество прямоугольников, все это может указывать на то, что дизайн неудачен. Таким образом, качество дизайна можно оценить на основании его графического представления.

Еще одним преимуществом визуального проектирования является скорость. За время, которое требуется для того, чтобы закодировать один вариант дизайна, вы можете сравнить три визуальных представления различных вариантов дизайна.

Недостатком графического представления является отсутствие надежной обратной связи. Имея перед глазами графическую схему системы, вы быстро получаете представление о том, насколько хорошо она спроектирована, и это в определенном смысле можно назвать полезной разновидностью обратной связи. Однако при этом вы лишаетесь другой разновидности обратной связи. К сожалению, обратная связь именно этой разновидности позволяет вам узнать о дизайне самое главное — можно ли с его помощью обеспечить срабатывание тестовых случаев? Позволяет ли данный дизайн обеспечить наиболее простую реализацию системы? Подобную обратную связь можно обеспечить только при помощи кодирования.

С одной стороны, если мы проектируем с использованием графики, мы можем делать это быстро. С другой стороны, проектируя с использованием графики, мы увеличиваем риск. Нам необходима стратегия, которая позволила бы воспользоваться преимуществами визуального проектирования и при этом нейтрализовать его недостатки.

К счастью, у нас есть все необходимое для разработки этой стратегии. У нас есть набор принципов, руководствуясь которыми мы можем действовать. Давайте взглянем:

- *Небольшие начальные инвестиции* — предполагает, что мы должны рисовать небольшое количество картинок за один раз.
- *Игра для победы* — предполагает, что мы должны использовать картинки не от собственного страха (например, для того, чтобы оправдать упущенный день, который мы тратим на решение проблем с дизайном).
- *Быстрая обратная связь* — предполагает, что мы должны быстро определить, приближают ли нас картинки к цели или нет.
- *Работать в соответствии с человеческими инстинктами* — предполагает, что мы ожидаем рисование картинок от тех, кому удобнее работать с картинками.
- *Принятие изменений и путешествие налегке* — предполагает, что мы не сохраняем картинки, которые уже оказали свое влияние на код, так как решения, которые иллюстрируются этими картинками, скорее всего, завтра потребуется изменить.

В рамках XP используется следующая стратегия: кто угодно может проектировать при помощи картинок что угодно, однако как только вста-

ет вопрос, ответ на который можно найти только при помощи кода, чтобы найти ответ, разработчики должны приступить к кодированию. Картинки не сохраняются. Например, графическую схему можно нарисовать на пластиковой доске фломастером. Если у вас возникает желание сохранить схему, это значит, что дизайн не был объяснен команде или не был отражен в системе.

Если вы имеете дело с разновидностью исходного кода, который лучше выражается при помощи картинок, тогда определенно вы должны выражать его, редактировать его и поддерживать его в виде картинок. Хорошим примером являются средства из категории CASE, которые позволяют вам целиком и полностью определять поведение всей системы при помощи графических изображений. Часто эту методику называют генерацией кода (code generation), или автоматической генерацией кода, однако для меня это — язык программирования. В этом случае я возражаю не против картинок, а против попыток хранения одной и той же информации о системе в двух разных синхронизированных между собой представлениях.

Если вы используете текстовый язык программирования, следуя этому совету, вы не должны тратить более чем 10–15 минут на рисование картинок. После этого вы поймете, какой вопрос вы хотите задать системе. После того как вы получите ответ, вы можете нарисовать еще несколько картинок до тех пор, пока вы не сформулируете еще один вопрос, который требует конкретного ответа.

Тот же совет имеет силу и в отношении других некодовых нотаций дизайна, таких как карты CRC. Занимайтесь этим в течение нескольких минут для того, чтобы сформулировать вопрос, затем обратитесь к системе, чтобы снизить риск того, что вы занимаетесь самообманом.

Системная архитектура

Я не использовал это слово ранее. На самом деле архитектура также важна для проектов XP, как и для любых других программных проектов. Частично архитектура выражается в системной метафоре. Если вы обладаете хорошей метафорой, каждый член команды может сказать, каким образом система работает как единое целое.

На следующем шаге необходимо увидеть, как именно история превращается в объекты. Правила игры в планирование предполагают, что в ходе первой итерации на свет должен появится функционирующий скелет системы как единого целого. Однако вы по-прежнему должны делать самую простую вещь, которая, возможно, сработает. Как можно удовлетворить оба этих условия?

Для первой итерации выберите набор простых, базовых историй, о которых можно предположить, что они позволят вам создать полностью всю

архитектуру. После этого ограничьте поле вашего зрения и реализуйте эти истории самым простым из всех возможных способов. После завершения этого упражнения вы получите архитектуру системы. Возможно, это не будет той архитектурой, которую вы ожидаете, однако в процессе работы вы лучше поймете, что именно вам необходимо.

Но что, если вы не можете подобрать набор историй, которые позволили бы вам сформировать архитектуру, которая вам необходима, в чем вы глубоко уверены? В этом случае вы можете либо сформировать архитектуру на основе только лишь размышлений и предположений, либо вы можете сформировать архитектуру так, чтобы решить ограниченный набор стоящих перед вами сейчас проблем, в надежде на то, что позже вы сможете развить имеющуюся архитектуру так, как это будет необходимо. Лично я предпочитаю формировать упрощенную архитектуру на основе стоящих передо мной задач, а затем при необходимости вносить в нее изменения.

Стратегия тестирования

Мы будем писать тесты перед тем, как приступить к кодированию. Это будет происходить каждый раз, когда мы будем садиться за решение очередной задачи. Мы сохраним эти тесты навечно и будем запускать их все вместе очень часто. Мы также будем писать тесты с точки зрения заказчика.

Какая жалость! Никто не хочет разговаривать о тестировании. Тестирование — это гадкий утенок индустрии разработки программного обеспечения. Проблема состоит в том, что каждый знает о важности тестирования. Каждый знает о том, что делает тестирование недостаточно хорошо. И мы видим это — наши проекты не реализуются так, как нам хотелось бы, и мы чувствуем, что тестирование в большем объеме может помочь решению проблемы. Но после этого мы беремся за чтение книги, посвященной тестированию, и увязаем в огромном количестве методик и разновидностей тестирования. Ни за что на свете нам не удастся выполнить все эти предписания и при этом завершить работу в срок.

В XP тестирование выглядит следующим образом. Каждый раз, когда программист пишет некоторый код, он думает, что этот код будет работать. Каждый раз, когда программист думает, что некоторый код будет работать, он берет свою уверенность из вселенского эфира и воплощает ее в артефакт, который становится частью программы. Теперь уверенность внутри, и программист может ею пользоваться. А так как уверенность теперь внутри программы, остальные люди могут также воспользоваться этой уверенностью.

То же самое можно сказать и о заказчике. Каждый раз, когда заказчик думает о чем-то конкретном, что должна делать программа, он превращает это в еще один кусок уверенности и размещает его внутри программы. Теперь уверенность заказчика тоже находится внутри программы рядом

с уверенностью программиста. Общая уверенность в работоспособности программы со временем все увеличивается и увеличивается.

Теперь вы можете взглянуть на тестирование в XP и хихикнуть: ведь это не работа для тех, кто любит тестирование, наоборот, это работа для тех, кто любит заставлять программы работать. Вы пишете тесты, которые помогают вам добиться работоспособности создаваемых вами программ, а также обеспечить работоспособность программ, которые вы модифицируете. И ничего больше.

Вспомните принцип: «Работать совместно с человеческой природой, а не вопреки ей». С этим связана фундаментальная ошибка, которая присутствует во всех книгах по тестированию, которые я прочитал. Все подобные книги начинаются с предположения, что тестирование — это центр разработки. Вы должны сделать этот тест и тот тест и, о да, конечно, вон тот тест тоже. Если мы хотим, чтобы программисты и заказчики писали тесты, мы должны сделать процесс разработки настолько безболезненным, насколько это возможно. Мы должны рассматривать тесты как инструмент разработки. Тесты — это инструмент, помогающий понять поведение системы, а поведение системы формируется разработчиками, а не самими тестами. Если бы было возможно осуществлять разработку без тестов, мы могли бы выбросить все тесты в одну минуту.

Массимо Арнольди (Massimo Arnoldi) пишет:

> К сожалению, по крайней мере для меня (и не только), тестирование — это нечто противоречащее человеческой природе. Где-то очень глубоко в каждом из нас сидит грязная неряшливая свинья, и если ее выпустить на свободу, мы будем писать программы совсем без тестов. Спустя некоторое время после этого у нас внутри победу одерживает рациональная сторона, мы останавливаемся и начинаем писать тесты. Следует обратить внимание, что программирование в паре снижает риск пробуждения свиньи, так как маловероятно, что в одно и то же время свиньи проснутся в обоих партнерах одновременно. (Источник: электронная почта.)

Тесты, которые необходимо писать в рамках XP, являются изолированными и автоматическими.

Во-первых, каждый тест никак не взаимодействует с остальными тестами, которые вы пишете. Таким способом вы избегаете ситуации, когда сбой в одном из тестов является причиной несрабатывания сотен других тестов. Ничто не разочаровывает нас так сильно, как ложные негативные результаты. Утром вы приходите на работу, обнаруживаете огромное количество дефектов, и у вас в кровь выделяется огромное количество адреналина. А потом обнаруживается, что весь этот ворох проблем решается при помощи коррекции одного из тестов. Будете ли вы относиться к тестам с должным вниманием после того, как описанное произойдет с вами пять или десять раз? Нет.

Во-вторых, тесты автоматические. Тесты наиболее полезны тогда, когда уровень стресса повышается, когда люди работают слишком много, когда человеческий здравый смысл начинает ослабевать. В подобных ситуациях было бы неплохо обладать быстрым и простым способом проверки работоспособности системы. Именно поэтому тесты работают автоматически — вы запускаете их и фактически сразу же получаете короткий односложный ответ: система работает корректно или система работает некорректно.

Протестировать абсолютно все невозможно — для этого тесты должны быть столь же сложными и столь же беззащитными перед ошибками, как и сам код приложения. Ничего не тестировать (в смысле изолированных автоматических тестов) — это самоубийство. Но тогда из всех вещей, которые можно протестировать, что именно вы должны тестировать?

Вы должны тестировать вещи, которые могут не сработать. Если код настолько прост, что он просто не может работать некорректно, и вы видите, что на практике данный код всегда срабатывает, тогда вы можете не писать для него код. Если бы я сказал вам, что надо тестировать абсолютно все, в скором времени вы пришли бы к выводу, что большая часть тестов, которые вы пишете, на самом деле бесполезна, и, если в этом отношении вы схожи со мной, вы перестали бы их писать. «Эти тесты предназначены для идиотов!»

Тестирование — это ставка. Ставка, которая оправдывает себя в случае, если оказывается, что ваши ожидания не соответствуют действительности. Тест может оправдать свое создание в ситуации, когда он срабатывает при том, что вы не ожидали, что он будет срабатывать. В этом случае вы должны выяснить, почему он срабатывает, так как код умнее, чем вы. Еще одна ситуация, в которой тест оправдывает свое создание, это когда тест не срабатывает, а вы ожидали, что он должен сработать. Очевидно, что в этом случае вы также должны разобраться, в чем дело. В обоих случаях вы узнаете кое-что новое о разрабатываемой вами программе. Разработка программного обеспечения — это получение новых знаний. Чем больше вы знаете, тем лучше вы разрабатываете код.

Таким образом, если у вас есть такая возможность, вы должны писать только те тесты, которые оправдывают затраты на свою разработку. Однако вы не можете заранее знать, какие из тестов оправданы, а какие — нет (если бы вы знали, это означало бы, что все необходимые знания у вас уже есть и вам не нужно узнавать ничего нового), поэтому вы должны разрабатывать тесты, которые *могут* оказаться оправданными. По мере того, как вы пишете тесты и выполняете тестирование, вы анализируете, какие разновидности тестов чаще оказываются оправданными, а какие — нет, и с течением времени вы начинаете писать все больше оправданных тестов и все меньше неоправданных.

Кто пишет тесты?

Как я уже говорил в самом начале главы, тесты возникают из двух источников:

- программисты;
- заказчики.

Программисты пишут тесты для каждого из методов. Тесты, разрабатываемые программистами, называются *тестами модулей* (unit test). Программист создает тест при следующих условиях:

- если интерфейс метода совершенно неясен, вы пишете сначала тест, а затем метод;
- если интерфейс ясен, однако вы полагаете, что при реализации могут возникнуть хотя бы и незначительные проблемы, вы пишете сначала тест, а затем метод;
- если вы обдумываете необычные условия, в которых код должен работать так, как это предполагается, вы должны написать тест для того, чтобы описать эти условия;
- если позже вы обнаруживаете проблему, вы должны написать тест, который изолирует эту проблему;
- если вы занимаетесь переработкой кода и вы не уверены в том, корректно ли он будет вести себя после переработки, и еще не существует теста, позволяющего проверить интересующий вас аспект поведения кода, вы должны вначале написать тест.

Написанные программистами тесты всегда срабатывают на 100%. Если один из тестов модулей не срабатывает, ни для одного члена команды не может быть другой более важной работы, чем исправление теста. На это может уйти минута. Но, может быть, для этого вам потребуется месяц. Вы не знаете заранее. И потому, что программисты контролируют написание и исполнение тестов модулей, они могут поддерживать все тесты в состоянии полной синхронизации с кодом системы.

Заказчики пишут тесты для каждой из историй. При этом они должны задавать себе вопрос: «Что еще должно быть проверено, прежде чем я буду уверен в том, что реализация этой истории полностью завершена?» Каждый создаваемый ими сценарий должен быть превращен в тест. В данном случае тесты называются *функциональными*.

Функциональные тесты не обязательно должны в любой момент времени срабатывать на все 100%. Дело в том, что эти тесты появляются из источника, который не является источником кода системы. По этой причине я не могу представить себе универсального способа синхронизации

функциональных тестов и кода системы в такой степени, в какой синхронизированы с кодом системы тесты модулей. В то время как для тестов модулей может быть только две оценки — 100% или ноль, работоспособность функциональных тестов, как правило, оценивается в процентном отношении. Ожидается, что спустя некоторое время все функциональные тесты должны срабатывать на все 100%. По мере приближения сроков выпуска очередной версии продукта, заказчик должен категоризировать не срабатывающие функциональные тесты. Некоторые из них являются для него более важными, чем другие, Исправление более важных функциональных тестов необходимо выполнять в первую очередь.

Как правило, заказчики не могут писать функциональные тесты самостоятельно. Они нуждаются в помощи того, кто сможет транслировать предоставляемые ими тестовые данные в собственно тесты, а также с течением времени разработать инструменты, при помощи которых заказчики могли бы писать, запускать и поддерживать свои собственные тесты самостоятельно. Именно поэтому команда XP любого размера должна включать в себя, по меньшей мере, одного программиста, основной обязанностью которого будет функциональное тестирование системы в тесном сотрудничестве с заказчиком. Его называют тестером (tester). Этот человек должен превращать временами весьма туманные идеи заказчика о функционировании системы в реальные, автоматические, изолированные тесты. Программист, занимающийся тестированием, также должен использовать разработанные с помощью заказчика тесты в качестве основы при создании разнообразных вариаций, которые могли бы указать на некорректное функционирование системы.

Даже если роль такого специалиста по функциональному тестированию играет человек, который получает удовольствие от взламывания программ, которые по идее должны быть готовы к использованию, этот человек должен работать в тех же экономических рамках, в которых работают и остальные программисты, занимающиеся разработкой тестов модулей. Иными словами, этот программист должен рассматривать каждый тест, как ставку в игре, надеясь на то, что тест сработает там, где ожидается его сбой, и на то, что тест не сработает там, где ожидается, что он должен сработать. Другими словами, этот программист должен писать только те тесты, разработка которых оправданна. Благодаря этому с течением времени он начинает производить все лучшие и лучшие тесты, тесты, которые с большей вероятностью оправдываются. Программист, занимающийся тестированием, существует вовсе не для того, чтобы создать как можно больше тестов. Его задача — создать тесты, которые лучше всего подчеркивают функциональность или, наоборот, недееспособность системы.

Другие тесты

Функциональные тесты и тесты модулей являются сердцем используемой в рамках XP стратегии тестирования, однако помимо них существуют также и другие тесты, использование которых может быть оправданно в определенных ситуациях. Команда XP должна проанализировать, в какой момент работы над проектом можно сбиться с пути, при этом необходимо определить, какие новые тесты могут оказаться полезными. Возможно, потребуется использовать следующие разновидности тестов (или любые другие тесты, описания которых можно найти в любой посвященной этому вопросу книге):

- *Параллельный тест* (parallel test) — этот тест предназначен для того, чтобы доказать, что новая система работает в точности, как старая система. На самом деле тест демонстрирует, насколько новая система отличается от старой. При этом заказчик может принять решение о том, насколько удовлетворительным для него является различие и допустима ли эксплуатация новой системы в промышленных условиях.
- *Стресс-тест* (stress test) — этот тест разрабатывается для того, чтобы сымитировать наиболее высокую нагрузку на систему. Стресс-тесты применяются для тестирования сложных систем, для которых сложно делать предположения о характеристиках, связанных с производительностью.
- *Тестобезьяны* (monkey test) — этот тест предназначен для того, чтобы продемонстрировать, что система корректно реагирует на бессмысленный, неподдерживаемый или запрещенный ввод.

Часть 3

Реализация XP

В данной части книги мы обсудим практическое применение стратегий, описанных в предыдущей части. После того, как вы выбрали упрощенный набор стратегий, вы получаете значительно большую гибкость, с которой вы можете их использовать. Вы можете использовать эту гибкость для многих целей, однако прежде всего вы должны знать о том, что эта гибкость существует, и о том, какие возможности она перед вами открывает.

Внедрение XP

Внедрять XP необходимо по одной методике за раз, всегда выбирая при этом наиболее серьезную проблему, которая стоит перед командой. Как только решаемая проблема перестает быть наиболее серьезной, вы переходите к следующей проблеме.

За простой и, очевидно, корректный ответ на вопрос о том, как следует внедрять XP, я хочу поблагодарить Дона Уэллса (Don Wells).

1. Выберите самую неприятную для вас проблему.
2. Решите ее, применяя способ XP.
3. Когда эта проблема перестает быть самой неприятной для вас, повторите эту последовательность действий с самого начала.

Две очевидные составляющие, с которых можно начать процесс внедрения, — это тестирование и игра в планирование. Очень многие проекты страдают от проблем, связанных с низким качеством кода, а также от дисбаланса полномочий между бизнесом и разработчиками. Вторая книга из серии книг, посвященных XP, под названием *Extreme Programming Applied: Playing to Win* («Применение экстремального программирования: игра чтобы победить») будет посвящена именно этим темам, так как с освоения именно этих компонентов XP удобнее всего приступать к внедрению этой дисциплины.

У описанного подхода к освоению XP существует большое количество преимуществ. Он настолько прост, что даже я могу понять его (после того, как Дон втолковал мне его суть). Вы овладеваете одной методикой за один раз, поэтому вы можете достаточно подробно изучить каждую из этих методик. Вы начинаете с решения наиболее актуальных для вас проблем, и поэтому у вас есть сильная мотивация, стимулирующая желание изменить все к лучшему, и, кроме того, вы немедленно получаете позитивную отдачу в ответ на ваши усилия.

Когда вы решаете наиболее мешающую вам проблему, вы также устраняете один из недостатков XP: «один размер предназначен для всех». Если вы занимаетесь изучением одной из входящих в XP методик, вы получаете возможность адаптировать ее для конкретно вашей ситуации. Если в некоторой области у вас нет никакой проблемы, вы даже и не подумаете о том, что в этой области XP можно использовать для решения каких-либо проблем.

Приступая к внедрению XP, не стоит недооценивать важность физического окружения, даже если ранее вы не имели никаких проблем в этой области. Лично я часто начинаю работу над проектом XP при помощи электрической отвертки и разводного ключа. Я также рекомендую добавить к описанному ранее процессу два дополнительных этапа:

-1. Переставьте мебель так, чтобы было удобно программировать парами и чтобы рядом с вами мог сесть заказчик.

0. Купите какую-нибудь закуску, например шоколадки, сухарики или крекеры.

20 Адаптация XP для существующего проекта

Проекты, в которых требуется изменить существующую культуру, встречаются гораздо **чаще,** чем проекты, в которых новую культуру необходимо сформировать с нуля. Внедряйте XP в рамках существующего проекта понемногу, начиная с тестирования или планирования.

Освоение XP с совершенно новой командой — это не так-то просто. Внедрение XP в уже существующих рабочих условиях, с существующей командой и существующим обширным кодом еще сложнее. Перед вами стоят все уже существующие у вас проблемы — обучение навыкам, инструктаж, обеспечение работы над проектом. Также вы обязаны обеспечить функционирование программного обеспечения, которое уже эксплуатируется в рабочих условиях. Это программное обеспечение разработано без учета ваших новых стандартов. Скорее всего, оно обладает значительно более сложной структурой, чем это требуется. Очевидно, оно не тестировалось в такой **степени,** в которой это требуется в рамках XP. Если вы собираетесь сформировать новую команду, вы можете выбрать тех людей, кто действительно хочет попробовать XP. Однако если вы имеете дело с уже существующей командой, возможно, в ней окажется несколько скептически настроенных членов. И вдобавок ко всем этим неприятностям все столы в рабочем помещении уже давно расставлены, и вы даже не имеете возможности организовать программирование парами.

Для адаптации XP в рамках существующего проекта вам потребуется больше времени, чем если бы вы занимались этим совместно с новой **командой,** приступая к работе над новым проектом. Это плохие **новости.** Однако есть и хорошие новости. Существуют риски, с которыми **вынуждены** столкнуться люди, которые организуют XP с нуля, и с которыми

вам не придется столкнуться. Вы не окажетесь в рискованной ситуации, в которой вы думаете, что у вас есть отличная идея программы, но со всей уверенностью вы об этом сказать не можете. Вы также не окажетесь в рискованной позиции, в которой вам будет необходимо делать множество решений без немедленной и жизненно-важной обратной связи с реальными заказчиками.

Я разговаривал со многими командами, которые говорили: «О, да! Мы уже работаем в рамках XP. У нас все, как в XP. Все, за исключением тестирования. Тестирование мы делаем по старинке. Да, и еще у нас есть 200-страничный документ, в котором изложены точные требования заказчика. Но все остальное мы делаем в точности как в XP». Именно поэтому данная глава состоит из нескольких разделов, каждый из которых соответствует одной из методик. Если вы уже внедрили у себя одну из методик, которая используется в рамках XP, вы можете игнорировать соответствующий раздел. Если вы намерены внедрить у себя какую-то новую методику, обратитесь к соответствующему разделу.

Каким образом можно внедрить XP с уже существующей командой и программным продуктом, который уже эксплуатируется на производстве? Вы должны модифицировать стратегию адаптации в следующих областях:

- тестирование;
- проектирование;
- планирование;
- менеджмент;
- разработка.

Тестирование

Когда вы приступите к приведению существующего кода в соответствие с требованиями XP, тестирование станет для вас, наверное, наиболее разочаровывающим процессом. Код, написанный до того, как вы пишете тесты, кажется пугающим. Вы никогда не знаете, где именно вы находитесь и в каком направлении без опасений можно сделать шаг. «Что, если я отредактирую эту строку? Будет ли это изменение безопасным?» Вы не уверены в этом.

Как только вы начинаете писать тесты, картина меняется. Вы уверены в новом коде. Вы не задумываетесь перед тем, как внести в него изменения. Для вас это становится даже приятным.

Переход от старого кода к новому коду — это как выход из темноты на солнечный свет. Вы начнете ловить себя на том, что вы избегаете рабо-

тать со старым кодом. Вы должны сопротивляться этой тенденции. Единственным способом обрести контроль над ситуацией в данном случае является переработка старого кода в соответствии с новыми более прогрессивными правилами. В противном случае в темноте начнут скапливаться страшные чудовища, которые в любой момент могут выпрыгнуть наружу. Оставляя старый код в прежнем состоянии, вы получаете риск, величину которого сложно оценить.

В подобной ситуации возникает соблазн вернуться несколько назад и написать тесты для всего существующего кода. Не делайте этого. Тесты для старого кода следует писать по мере надобности.

- Если вы хотите добавить новую функциональность в не тестированный код, вначале напишите тесты для существующей функциональности.
- Если вы намерены исправить ошибку в старом коде, сначала напишите тест.
- Если вы намерены переработать участок старого кода, сначала напишите все необходимые тесты.

Вы обнаружите, что вначале разработка несколько замедлится. Вы будете тратить существенно больше времени на написание тестов, чем требуется для этого в рамках обычной XP, и у вас появится ощущение, что вы формируете новую функциональность более медленно, чем раньше. Однако разделы системы, к которым вы обращаетесь чаще всего, части, которые привлекают к себе наибольшее внимание, а также новые возможности системы в обозримом будущем будут тщательно протестированы. В скором времени части системы, использующиеся чаще других, будут выглядеть, как будто они с самого начала написаны с применением XP.

Проектирование

Переход к проектированию в стиле XP во многом напоминает переход к тестированию в стиле XP. Вы обнаружите, что по сравнению со старым кодом, новый код выглядит совершенно по-другому. Вам захочется исправить все сразу. Не делайте этого. Модернизацию следует осуществлять постепенно. По мере добавления новой функциональности будьте готовы перед этим выполнить переработку кода. Когда вы разрабатываете программу в рамках XP, вы всегда готовы вначале выполнить переработку, однако если вы осуществляете переход на XP существующего проекта, вам придется выполнять переработку чаще.

На ранних стадиях процесса команда должна определить долгосрочные цели переработки существующего кода. Возможно, в проекте исполь-

зуется слишком запутанная иерархия наследования классов, возможно, некоторая важная функциональность разбросана по всей системе и вы желаете собрать ее воедино. Сформулируйте все эти цели, запишите их на карточках и развесьте эти карточки на видных местах. Когда вы сможете сказать, что большая переработка закончена (для этого могут потребоваться месяцы или даже год работы в описанном «постепенном» стиле), можете устроить посвященную этому веселую вечеринку. Торжественно сожгите карточки. Хорошенько выпейте и закусите.

Эффект этой стратегии во многом напоминает эффект стратегии тестирования по необходимости. Те части системы, к которым вы обращаетесь чаще всего, в скором времени будут напоминать код, изначально разрабатываемый с применением принципов XP. Дополнительная нагрузка, связанная с необходимостью переработки существующего кода, в скором времени растворится в воздухе.

Планирование

Вы должны преобразовать существующую информацию о требованиях в набор карточек с историями. Вы должны обучить вашего заказчика правилам игры. Заказчик должен решить, что необходимо включить в следующую версию программы.

Наибольшая сложность (равно, как и преимущество) при переходе к планированию в стиле XP состоит в том, что вы должны объяснить вашему заказчику, насколько больше он сможет получить от команды, если перейдет на новые правила взаимодействия с разработчиками. Скорее всего, заказчик ранее не имел опыта работы с командой, которая приветствует внесение изменений в требования. Конечно, для того, чтобы привыкнуть к новым открывающимся перед ним возможностям, заказчику потребуется некоторое время.

Менеджмент

Освоение менеджмента XP — один из наиболее сложных переходов в процессе адаптации к XP. Менеджмент XP — это игра в направление и влияние. Если вы менеджер, скорее всего, вы поймаете себя на том, что принимаете решения, которые должны приниматься либо программистами, либо заказчиками. Если этот так, то не паникуйте. Просто напомните себе и всем присутствующим, что вы просто обучаетесь. После этого попросите нужного человека принять решение и поставить вас в известность о том, что решено.

Программисты, неожиданно наделенные новой ответственностью, вряд ли сразу же начнут делать большую работу. Как менеджер, в течение пе-

реходного периода вы должны с особенной тщательностью напоминать всем об избранных правилах действий. В состоянии стресса каждый будет пытаться вернуться к старому стилю поведения вне зависимости от того, эффективным ли был этот стиль или нет.

Ощущения будут напоминать те, которые возникают при адаптации к XP проектированию или тестированию. Поначалу все будет казаться вам неудобным. Вы будете ощущать, что вы действуете не с самой быстрой возможной скоростью. Однако если вы будете уделять время изучению ситуаций, возникающих изо дня в день, то вы (равно как и программисты, и заказчики) научитесь решать все возникающие проблемы гладко. В скором времени вы почувствуете себя комфортно в новом процессе. Однако время от времени будет возникать ситуация, в которой вы будете сталкиваться с тем, что сделано до того, как вы приступили к внедрению XP. Как только возникнет такая ситуация, сделайте шаг назад. Напомните команде о правилах, ценностях и принципах. После этого решите, что делать.

Наиболее сложным аспектом менеджмента в процессе стремительного перехода к использованию XP является принятие решения о том, что некоторый член команды не справляется с работой в новых условиях. В подобной ситуации лучше расстаться с ним. При этом, если вы уверены, что ситуация не изменится в лучшую сторону, вы должны сделать изменения так быстро, как это возможно.

Разработка

Первым делом, которое вы обязаны сделать, является проверка расположения столов. Я не шучу. Прочтите заново материал о программировании парами (см. главу 16). Передвиньте ваши столы так, чтобы за каждым из них могли с удобством расположиться два человека и чтобы они могли передавать друг другу клавиатуру без необходимости двигать при этом стулья.

С одной стороны, при переходе к использованию XP вы должны более строго следить за обязательным использованием программирования парами. Поначалу программирование парами может показаться вам неудобным. Вы должны заставлять себя программировать парами, даже если вам кажется, что это неудобно и неэффективно. С другой стороны, вы должны время от времени делать перерывы. Уединитесь на пару часиков и попрограммируйте в одиночку. Конечно же, результаты вашей работы следует выбросить, но вы ни в коем случае не должны отказываться от удовольствия, которое вы получаете от программирования, только для того, чтобы сказать, что вы программируете в парах в течение 30 часов за одну неделю.

Проблемы?

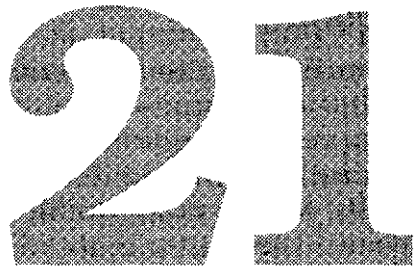
Некоторые из читателей данной книги уже обладают готовыми сформированными командами, однако разрабатываемые ими программные системы еще не поступили в эксплуатацию. Возможно, ваш проект погряз во множестве проблем и XP может выглядеть как возможное спасение.

Не рассчитывайте на это. Возможно, если бы вы использовали XP с самого начала, вы смогли бы (или не смогли бы) избежать возникновения текущей ситуации. Однако если менять коня на переправе сложно, то менять умирающего коня в десять раз сложнее. Эмоции будут на пределе. Моральный дух будет очень низким.

Если перед вами стоит выбор: либо использовать XP, либо быть уволенным, прежде всего поймите, что ваши шансы последовательно и тщательно внедрить новые методики работы очень низки. В состоянии стресса вы будете постоянно возвращаться к старым привычкам. У вас и без того хватает напряжения. Вы пытаетесь сделать работу, которая ускользает у вас из рук. К этому вы хотите добавить дополнительную нагрузку, связанную с внедрением XP. Таким образом, ваши шансы на успешный переход к XP существенно снижаются. Выберите для себя более скромную цель, чем спасение всего проекта. Приближайтесь к ней постепенно, день за днем. Довольствуйтесь тем, что нового вы можете узнать о тестировании или управлении проектом не напрямую, или насколько красивым вы можете сделать дизайн, или насколько большой объем кода вы можете удалить. Возможно, если вы не будете думать о завтрашнем дне, из хаоса сформируется более приемлемый для вас порядок.

Все же если вы собрались переводить проблемный проект на использование XP, сделайте это драматическим событием. Полумеры, скорее всего, не помогут — все останется в более-менее прежнем состоянии. Тщательно проанализируйте весь имеющийся код. Сможете ли вы обойтись без него? Может быть, будет лучше забыть о его существовании? Если да, то выбросите его. Целиком. Разожгите огромный костер и торжественно сожгите старые ленты резервного копирования. Сделайте неделю передышки. И после этого начните все заново со свежими силами.

Жизненный цикл идеального XP-проекта



Идеальный проект XP проходит сквозь короткую стадию начальной разработки, за которой следуют годы поддержки эксплуатации системы на производстве и одновременно пересмотра и переделки. Наконец, когда проект теряет актуальность, он элегантно отправляется в отставку.

В данной главе дается представление о полном жизненном цикле проекта XP — его истории от начала до конца. История идеализирована — я полагаю, что на текущий момент вы уже поняли, что никаких два проекта XP не могут (да и не должны) быть абсолютно одинаковыми. В данной главе дается представление о фазах жизни проекта.

Исследование

Предварительная подготовка к работе — это неестественное состояние системы, и этот этап должен быть завершен как можно быстрее. Какую фразу я услышал совсем недавно? «Если программа начинает эксплуатироваться на производстве, значит, программа завершена». XP утверждает прямо противоположное. Если программа еще не эксплуатируется на производстве, это значит, что мы тратим деньги и при этом не зарабатываем деньги. Я рассматриваю эту ситуацию так, как будто это мой бумажник. Ситуация, когда деньги тратятся и при этом не зарабатываются, кажется мне очень дискомфортной.

Однако прежде, чем вы сможете приступить к эксплуатации системы, вы должны поверить в то, что система может эксплуатироваться. Вы должны убедиться в том, что имеющиеся у вас инструменты позволят вам успешно завершить работу над программой. Вы должны поверить в то, что, когда код завершен, вы сможете запускать его изо дня в день. Вы должны

поверить в то, что у вас есть (или вы сможете получить) все необходимые навыки, которые вам потребуются для работы. Члены команды должны научиться доверять друг другу

Именно в ходе фазы исследования все эти составляющие собираются в единое целое. Исследование можно считать завершенным тогда, когда заказчик уверен в том, что на карточках историй содержится более чем достаточно материала для того, чтобы сделать самую первую хорошую версию системы, а программисты уверены в том, что они уже не могут оценить эти истории лучше, не реализовав при этом систему в коде.

В процессе исследования программисты осваивают каждую из составляющих технологии, которую они собираются применять при разработке системы. Они активно исследуют варианты организации системной архитектуры. Для этого в течение одной или двух недель они формируют систему, подобную той, которую они собираются реализовать, но несколькими разными способами. Несколько разных пар могут попробовать сформировать систему несколькими разными способами, а затем сравнить. Возможен и другой подход. Две пары по отдельности реализуют систему одним и тем же способом, затем анализируют получившиеся в результате различия.

Если недели недостаточно для того, чтобы освоиться с некоторой технологией, эту технологию следует классифицировать как рискованную. Это не означает, что вы не можете ее использовать. Однако вы должны изучить ее более тщательно и рассмотреть возможные альтернативы.

Во время исследования вы можете привлечь к работе сторонних специалистов, владеющих некоторой технологией лучше, чем вы. Благодаря сторонней помощи вам не придется сражаться с какими-либо проблемами, которые могут быть решены без затруднений, если только знать правильный подход. Как правило, специалисты владеют множеством таких правильных подходов. Однако относиться к советам специалистов необходимо с должной осторожностью. Не следует слепо доверять всему, что они говорят. Дело в том, что у экспертов часто вырабатываются привычки, основанные на применении некоторой технологии в крупномасштабных системах. Это не всегда то, на что вы рассчитываете, и это не всегда хорошо сочетается с принципами экстремального программирования. Команда должна хорошо владеть избранными методиками. Фраза: «Так посоветовал эксперт» не является удовлетворительным аргументом в случае, если проект выходит из-под контроля.

Программисты должны также экспериментировать с пределами производительности для технологии, которую они собираются использовать. Если это возможно, они должны имитировать реалистичный уровень на-

грузки на используемое в производстве аппаратное обеспечение и сеть. Это не означает, что вы должны в лабораторных условиях воспроизвести копию промышленной аппаратной платформы. Очень многие оценки можно сделать на основании расчетов. Например, прикинув примерный объем данных, передаваемых в рамках одного запроса, вы сможете вычислить, какой пропускной способностью должен обладать канал связи. После этого вы можете провести эксперимент, чтобы увидеть, возможно ли реализовать это на практике.

Программисты должны также экспериментировать с архитектурными идеями — как построить систему с несколькими уровнями отмены действий? Попробуйте реализовать этот механизм тремя разными способами и определите, какой из них самый эффективный. Подобные небольшие исследования в области архитектуры особенно важны в случае, если к вам приходит пользователь системы и рассказывает вам истории, которые вы не знаете, как реализовать.

Программисты должны оценить каждую из задач, которые им придется решить в процессе исследований. Проводя исследования, они должны прикинуть, какое время потребуется им для решения задачи. Когда работа над задачей завершается, необходимо сравнить предварительно сделанную оценку с реальным календарным временем, которое потребовалось для решения задачи. Использование такого подхода на практике повышает уверенность команды в своих оценках.

Пока команда практикуется с технологиями, заказчик практикуется с историями. Не ждите, что этот процесс будет протекать гладко и без запинок. Поначалу истории будут не такими, как вам хотелось бы. Чтобы решить проблему, необходимо обеспечить заказчика быстрой обратной связью. Как можно быстрее сообщайте ему вашу оценку предоставляемых им историй. Вы должны сообщить ему, что правильно, а что — нет. Какие данные должны указываться в истории, а какие — нет. В скором времени заказчик научится включать в историю именно то, что нужно программистам, и не включать в нее то, в чем программисты не нуждаются. Ключевым является вопрос: «Могут ли программисты с уверенностью оценить усилия, которые потребуются для реализации истории?» Иногда оказывается, что историю требуется реализовать иначе. Иногда оказывается, что программисты должны на некоторое время приостановить свое продвижение вперед и заняться экспериментированием.

Если у вас в распоряжении команда, члены которой хорошо знают друг друга, а также уже владеют технологией, которую предполагается использовать, фаза исследований может занять не более нескольких недель. Если команда плохо знакома с технологией, а ее члены еще не успели сработаться друг с другом, для исследований может потребоваться несколько

месяцев. Если исследование требует еще больше времени, я порекомендовал бы заняться реализацией менее крупного, но более реального проекта, с которым команда справится без затруднений. В процессе работы вы сможете лучше овладеть дисциплиной и освоиться с методиками.

Планирование

В ходе фазы планирования заказчики и программисты должны прийти к обоюдоодобренному соглашению по поводу даты, к которой будет реализован наименьший возможный полезный для бизнеса набор историй. Этот процесс описан в разделах, посвященных игре в планирование. Если в ходе исследовательской фазы вы достаточно хорошо подготовились, фаза планирования (формирование графика работ) не займет у вас больше, чем один-два дня.

План работ над первой версией продукта должен быть рассчитан на срок от двух до шести месяцев. За более короткое время вы вряд ли сможете решить какие-либо значительные проблемы бизнеса. (Однако если вы в состоянии справиться за более короткое время, это просто замечательно! В книге Тома Гилба (Tom Gilb) под названием *Principles of Software Engineering Management* «Принципы менеджмента в области разработки программного обеспечения» содержатся идеи, направленные на сокращение даты выхода первой версии продукта.) Если для работы над первой версией требуется больше времени, значит, риск разработки становится слишком большим.

Итерации в первой версии

График работы над первой версией разбит на несколько итераций длительностью от одной до четырех недель. В ходе каждой итерации разрабатывается набор функциональных тестовых случаев для каждой из историй, запланированных для выпуска в рамках данной итерации.

В ходе первой итерации происходит формирование архитектуры. По этой причине для первой итерации следует подобрать истории, которые стимулируют формирование «общей структуры» системы, пусть даже в форме примитивного каркаса, к которому в дальнейшем будет пристыковываться вся остальная функциональность.

Подбор историй для последующих итераций целиком и полностью осуществляется на усмотрение заказчика. При этом заказчик должен задать себе вопрос: «Какая возможность системы является для нас наиболее полезной?» Именно самые полезные возможности включаются в состав следующей итерации.

Занимаясь реализацией итераций, вы следите за отклонением от плана. Потребовалось ли для реализации чего-либо в два раза больше време-

ни, чем вы думали? Может быть, вы управились в два раза быстрее? Реализованы ли тестовые случаи в срок? Насколько приятно вам работать?

Если вы обнаруживаете отклонения от плана, значит, вы должны чего-то изменить. Возможно, требуется изменить план — добавить или убрать истории или изменить объем работ. Может быть, изменения следует внести в процесс — вы обнаружили более эффективные методы использования вашей технологии или более эффективные способы внедрения XP.

В идеале, в конце каждой итерации у заказчика есть набор завершенных функциональных тестов и все они срабатывают. В конце каждой итерации рекомендуется устроить небольшую праздничную церемонию — купите пиццу, запалите пару фейерверков, пусть заказчик оставит свой автограф на карточках реализованных историй. Еще бы, ведь вы разработали качественный программный продукт в срок! Возможно, для этого вам потребовалось лишь три недели, но все равно это достижение и это стоит отметить!

В конце последней итерации вы готовы приступить к эксплуатации системы в реальных производственных условиях.

Внедрение в эксплуатацию

На завершающих стадиях работы над очередной версией следует сократить цикл обратной связи. Вместо трехнедельных итераций необходимо перейти на итерации продолжительностью в одну неделю. Возможно, будет полезным устраивать ежедневное совещание, благодаря чему вся команда будет точно знать, над чем в данный момент работает тот или иной член команды.

Как правило, для сертификации программного обеспечения (то есть для того, чтобы определить, готово ли оно к использованию в реальных производственных условиях) используется специальный процесс. Будьте готовы к тому, что вам потребуется разработать новые тесты для того, чтобы убедиться в готовности системы к реальной работе. Зачастую именно на этой стадии применяется параллельное тестирование.

В ходе этой фазы вы также оптимизируете производительность системы. В данной книге я почти ничего не говорю о производительности. Я уверен в лозунге: «Сделайте, чтобы это заработало, сделайте, чтобы это было написано правильно, сделайте, чтобы это работало быстро» (Make it run, make it right, make it fast). На завершающих стадиях работы над версией наступает отличное время для оптимизации, потому что именно в это время вы получаете максимально возможное количество знаний, внедренных в дизайн системы, вы получаете наиболее реалистичные оценки производственной нагрузки на систему и, кроме того,

вы скорее всего получаете в свое распоряжение реальную производственную аппаратную платформу.

В процессе внедрения очередной версии системы в эксплуатацию вы замедляете темпы эволюционирования системы. Это не означает, что программа вообще перестает эволюционировать, скорее риск становится более значимым фактором, когда вы размышляете, заслуживает ли то или иное изменение того, чтобы включить его в систему. Однако имейте в виду, что чем большим опытом работы с системой вы будете обладать, тем более четко вы представляете себе, как она должна быть спроектирована. Если у вас возникает огромное количество идей, которые вы не можете включить в текущую версию системы, составьте список и сделайте его доступным для обозрения. Благодаря этому каждый сможет увидеть, в каком направлении будет развиваться система после того, как текущая версия начнет эксплуатироваться на предприятии заказчика.

Когда очередная версия программного продукта будет внедрена, устройте большой праздник. Многие проекты никогда не доходят до внедрения, поэтому если ваш проект дошел до этой стадии и продолжает жить, — это отличная причина собраться и отметить очередную вашу победу. Если на этой стадии вы не чувствуете никаких, пусть даже легких, опасений, значит либо у вас железные нервы, либо вы сумасшедший, однако веселая вечеринка поможет вам избавиться от напряжения, которое в противном случае может только возрасти.

Обслуживание и поддержка

Обслуживание и поддержка в действительности — это нормальное состояние любого XP-проекта. Вы должны одновременно работать над реализацией новой функциональности, поддерживать существующую систему в рабочем состоянии, принимать в команду новых членов и говорить слова прощания тем, кто уходит.

Работа над каждой очередной версией начинается с исследований. Вы можете попробовать выполнить крупномасштабную переработку кода, которой вы опасались на завершающих стадиях работы над предыдущей версией. Вы можете попробовать использовать новую технологию, поддержку которой вы намеревались обеспечить в очередной версии продукта. Возможно, вы захотите перейти на использование новой версии технологии, которую вы уже применяете в рамках продукта. Вы можете поэкспериментировать с новыми архитектурными идеями. Заказчик может попытаться написать новые бредовые истории в поисках золотой жилы, которая способна многократно увеличить производительность бизнеса.

Разработка системы, которая уже функционирует в условиях реального производства, — это не одно и то же, что и разработка системы, кото-

рая еще не эксплуатируется на реальном предприятии. Вы более осторожно относитесь к любым осуществляемым вами изменениям. Вы должны быть готовы прервать дальнейшую разработку для того, чтобы прореагировать на проблемы, которые возникли на производстве. Вы имеете дело с реальными данными, с которыми надо обходиться чрезвычайно осторожно. Вы должны позаботиться о миграции этих данных при любом в достаточной степени значительном изменении дизайна. Если бы фаза предварительной разработки не была бы столь рискованной и опасной, вы могли бы оттягивать момент внедрения системы неограниченно долгое время.

Внедрение системы в производство, скорее всего, повлияет на скорость разработки. Делая новые оценки, будьте более консервативны.

В процессе исследований оцените эффект, который окажет на процесс разработки необходимость поддержки функционирования системы в реальных производственных условиях. После внедрения первой версии системы на производстве я наблюдал существенные изменения отношения идеального времени разработки к реальному календарному времени (с двух календарных дней к одному идеальному до внедрения до трех календарных дней к одному идеальному после внедрения). Не стоит делать туманных предположений. Постарайтесь определить этот коэффициент точно.

Будьте готовы изменить структуру команды специально для того, чтобы эффективнее обслуживать функционирование системы. Возможно, вам захочется организовать нечто вроде службы технической поддержки, чтобы большая часть программистов не отрывалась слишком часто от текущей разработки для решения возникающих производственных проблем. Следует организовать смену персонала в этой службе таким образом, чтобы со временем все работающие над системой программисты побывали в этой роли, — существуют вещи, которым можно научиться, только осуществляя техническую поддержку системы на производстве и о которых нельзя узнать как-либо иначе. С другой стороны, техническая поддержка — занятие менее веселое, чем разработка.

Размещайте новые разработанные куски программы в работающей на производстве системе по мере их разработки. Возможно, вплоть до очередной версии эти части не будут использоваться и не будут работать. Я все равно рекомендую вам добавлять новый разработанный код в реально работающую систему. Я работал над проектами, в которых подобный цикл выполнялся ежедневно или еженедельно. В любом случае, вы не должны оставлять готовый код в бездействующем состоянии дольше, чем в течение одной итерации. Это время определяется величиной затрат, связанных с верификацией кода и миграцией данных. Последнее действие, которое вам будет необходимо выполнить в конце работы над версией,

это интеграция большого куска кода, который, скорее всего, не сможет ничего поломать. Если вы будете поддерживать код, используемый на производстве, и код, находящийся в разработке, приблизительно в синхронизированном состоянии, вы будете раньше узнавать об интеграционных проблемах.

Когда в команде появляются новые люди, предоставляйте им две или три итерации, в течение которых они задают массу вопросов, выполняют роль партнеров при программировании в парах и изучают огромные объемы тестов и кода. Когда они почувствуют себя достаточно подготовленными, они смогут принять на себя ответственность за выполнение новых задач, однако фактор нагрузки для них необходимо снизить. Когда они продемонстрируют свою способность выпускать качественный код, фактор нагрузки можно будет поднять.

Если состав команды будет меняться постепенно, меньше чем за год вы можете заменить изначальную команду абсолютно новыми людьми, не нарушая при этом как технической поддержки работающего продукта, так и текущей разработки новой функциональности. Это значительно менее рискованный подход, чем типичное: «Вот эта и эта кипы бумаги содержат всю необходимую для вас информацию, изучив которую вы сможете приступить к работе». В действительности передать новичку информацию о культуре, в рамках которой ведется разработка проекта, — это также важно, как передать информацию о деталях дизайна и реализации, и это возможно только при личном контакте.

Смерть

Хорошо умереть — это также важно, как и хорошо жить. Для XP это является такой же истиной, как и для людей.

Если заказчик больше не может придумать ни одной новой истории, значит, наступило время хорошенько посыпать систему нафталином. Теперь необходимо написать пяти-, может быть, десятистраничное описание системы — документ, который может вам потребоваться в случае, если через пять лет вы захотите что-то изменить внутри.

Это хорошая причина для того, чтобы умереть, — заказчик доволен системой и не может придумать ничего, что можно было бы добавить в систему в обозримом будущем (я никогда с таким не сталкивался, однако я слышал об этом, поэтому я рассматриваю этот вариант для полноты картины).

Существует также и другая, не очень хорошая причина для смерти — система находится в неудовлетворительном состоянии. Заказчик нуждается в новых возможностях, а вы не можете добавить их по экономиче-

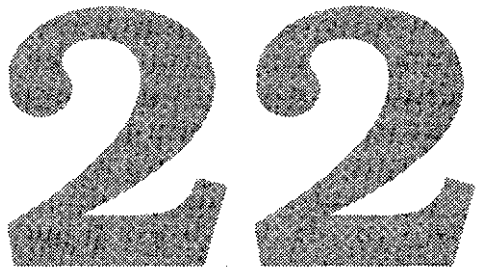
ским соображениям. Количество дефектов может вырасти до величины, которая является неприемлемой.

Это смерть от энтропии, с которой вы боретесь как можно более долгое время. XP — это не волшебство. Проекты XP подвержены энтропии точно так же, как и любые другие проекты. Вы просто надеетесь, что это произойдет как можно позже.

В любом случае, мы столкнулись с невозможным — система должна умереть. Когда это происходит, глаза всех участников проекта должны быть открыты. Команда должна быть в курсе экономической ситуации. Команда, заказчики и менеджеры должны согласиться с тем, что ни команда, ни система не могут выдать заказчику то, что ему нужно.

Теперь наступает время нежного прощания. Устройте вечеринку. Пригласите всех, кто работал над проектом. Это должен быть вечер воспоминаний. Воспользуйтесь возможностью и попытайтесь проанализировать причины того, что система погибла. Благодаря этому вы сможете лучше понять, с чем вам, возможно, придется иметь дело в будущем. Вместе с командой подумайте о том, как именно следует изменить свои действия, чтобы добиться успеха при работе над следующим проектом.

Роли для людей



Для того чтобы команда XP могла работать, необходимо, чтобы кто-то взял на себя исполнение некоторых определенных ролей: программиста, **заказчика,** инструктора, ревизора.

Спортивная команда играет лучше, если каждый игрок берет на себя исполнение определенной роли. Например, в состав футбольной команды, как правило, входит вратарь, нападающий, защитник и так далее. В баскетбольной команде присутствуют центр, нападение, защита и так далее.

Игрок, играющий в одной из этих позиций, принимает на себя определенный набор обязанностей — помощь своим соратникам в нападении, защита от нападения соперников, возможно, контроль над определенным участком поля. Некоторые из этих ролей предусматривают действия в индивидуальном порядке. Другие созданы для исправления ошибок других игроков команды и координации их действий.

Подбор этих ролей основан на опыте и зачастую определяется в правилах игры. Это сделано потому, что данная комбинация ролей признана наиболее эффективной. Возможно, когда-то давно люди пробовали другие комбинации распределения обязанностей между игроками. Но то, что мы видим сегодня, сохранилось благодаря тому, что именно такая комбинация оказалась более эффективной, чем другие.

Хорошие тренеры учат игроков хорошо исполнять возложенные на них обязанности, иными словами, хорошо играть свою роль. Тренер наблюдает за поведением игрока, указывает на отклонения и либо помогает игроку скорректировать свое поведение, либо объясняет, почему игрок может вести себя несколько иначе.

Однако самый лучший тренер знает, что распределение ролей основано на опыте и не является неизменным законом природы. Время от времени

игра меняется или игроки изменяются настолько, что появляется возможность добавить в существующий расклад новые роли или удалить из команды устаревшие, ставшие ненужными позиции. Гениальный тренер всегда следит за тем, какие преимущества появятся у команды в случае, если добавить новую роль или избавиться от одной из существующих.

Еще одна способность, которой обладают великие спортивные тренеры, — это их способность формировать систему в соответствии с командой, а не наоборот. Если у вас есть система, которая работает превосходно в случае, если вы используете быстрых игроков, и если команда, с которой вы работаете, показала себя на тренировках малоподвижной, но взамен этого сильной и выносливой, для вас будет лучше разработать новую систему, которая позволила бы вашей команде в полной мере проявить присущие ей таланты. Большинство тренеров не владеют этим. Вместо этого они настолько фокусируются на красоте системы, что подчас не замечают, что она не срабатывает.

Все это должно служить большим предупреждением для тех, кто слишком сильно привыкает к условиям, которые кажутся ему идеальными. Существуют роли, которые хорошо работали в предыдущих проектах. Если сейчас у вас в распоряжении люди, которые не соответствуют этим ролям, измените роли. Не пытайтесь менять людей (по крайней мере слишком значительно). Не следует действовать так, как будто никакой проблемы нет. Если роль предписывает, что «этот человек должен быть склонен к большому риску», и вместо этого вы имеете дело со скрупулезным дотошным человеком, который все привык рассчитывать заранее и не предпринимает никаких действий, не взвесив все «за» и все «против», вы должны найти иное разделение обязанностей, которое будет соответствовать поставленной перед вами цели, но без этой самой роли, которую должен играть рисковый человек.

Например, в свое время я общался с одним менеджером. Разговор шел о его команде. В ней программист являлся также и заказчиком. Я сказал моему собеседнику, что это не может эффективно работать, так как программист должен исполнять процесс и принимать технические решения. При этом необходимо, чтобы бизнес-решения принимались отдельно — заказчиком (см. описание игры в планирование).

Менеджер стал со мной спорить: «Этот парень реальный биржевой брокер, выяснилось, что он умеет также и программировать. Другие брокеры любят и уважают его. Они уверены в нем и желают видеть его в качестве своего доверителя. Он обладает четким представлением о том, в каком направлении должна развиваться система. Другие программисты хорошо различают, когда он разговаривает от лица заказчика и когда он разговаривает от лица технического специалиста».

Хорошо. Правила утверждают, что программист не может быть заказчиком. В данном случае эти правила не применяются. Но по-прежнему остается в силе принцип разделения технических решений и бизнес-решений. Вся команда, сам программист/заказчик, и особенно инструктор XP, должны точно знать, какую роль играет программист/заказчик в каждый конкретный момент времени. Кроме того, инструктор должен знать, что вне зависимости от того, насколько хорошо данная организация работала в прошлом, если команда попадает в затруднительную ситуацию, двойная роль является наиболее вероятной причиной проблемы.

Программист

Программист является сердцем XP. На самом деле если бы программисты могли всегда принимать решения, в которых тщательно балансировались краткосрочные и долгосрочные приоритеты, в рамках проекта не нужны были бы никакие другие технические работники, кроме программистов. Конечно же если заказчику не требуется программное обеспечение для того, чтобы поддерживать функционирование бизнеса, то никакой надобности в программистах не было бы.

С первого взгляда может показаться, что быть программистом XP — это то же самое, что быть программистом в любом другом проекте. Вы проводите свое время, работая над программами, увеличивая их размер, упрощая код и повышая производительность приложений. Однако если копнуть глубже, в XP ваши усилия концентрируются несколько иначе, чем в других дисциплинах. Ваша работа не заканчивается тогда, когда компьютер начинает вас понимать. Вашей основной задачей является передача важных сведений другим людям. Если программа уже работает, но при этом некий важный компонент коммуникации не завершен, значит, вы должны продолжить работу. Вы пишете тесты, которые демонстрируют жизненно важные аспекты разрабатываемого вами программного продукта. Вы разделяете программу на меньшие фрагменты или объединяете слишком маленькие куски программы в более крупные, более емкие части. Вы подыскиваете систему именования таким образом, чтобы выбираемые вами имена более точно отражали ваши намерения.

Это может показаться благородным стремлением к совершенству. На самом деле это все что угодно, но не погоня за совершенством. Вы пытаетесь разработать программное обеспечение, которое будет как можно более полезным для заказчика, однако при этом вы не пытаетесь делать чего-либо, что не является полезным. Если вы в достаточной степени сократите масштаб проблемы, тогда вы можете позволить себе более тщательно работать над тем, что от нее осталось. В результате вы начинаете работать тщательно по привычке.

Как программист XP вы должны обладать навыками, которые не требуются или по крайней мере не являются настолько важными в рамках других стилей разработки. Например, программирование в паре — это не такое уж и сложное искусство, им вполне можно овладеть, однако зачастую оно вступает в противоречие с некоторыми особенностями людей, которые, как правило, занимаются программированием. Подозреваю, что я должен выразить свою мысль более простым языком: многие программисты являются малообщительными людьми. Конечно, из этого правила есть исключения, кроме того, если ты не чувствуешь себя свободно в общении, этому не сложно научиться. Однако факт остается фактом: для того, чтобы успешно делать свою работу, вы должны близко общаться с другими членами команды и координировать с ними свою деятельность.

Еще одним навыком, который необходим программисту XP, является привычка к простоте. Когда заказчик говорит: «Вы должны сделать это, это и это», вы должны быть готовым к обсуждению, являются ли все эти вещи действительно необходимыми и насколько? Простота также должна распространяться на разрабатываемый вами код. Программист, который привык в каждой из возможных ситуаций использовать то или иное готовое решение, вряд ли сможет удачно действовать в рамках XP. Конечно же, скорее всего, вы справитесь со своей работой лучше, если у вас в запасе будет большее количество инструментов и приемов, однако будет лучше, если у вас будет множество инструментов, о которых вы будете знать, когда их *не* следует использовать. Это будет лучше, чем если бы вы знали все обо всем и с риском использовали бы в своем решении большое количество разнообразных приемов.

Вам также потребуются навыки, в большей степени технически ориентированные. Вы должны уметь хорошо программировать. Вы должны уметь перерабатывать код — это умение по крайней мере столь же глубокое и тонкое, сколь и программирование. Вы должны уметь писать тесты модулей для вашего кода — это умение, как и переработка, требует вкуса и рассудительности.

Вы должны отвыкнуть от желания получить в индивидуальное владение некоторую часть системы. Напротив, вы должны рассматривать всю систему как общую собственность, которая принадлежит не только вам, но и вашим товарищам по команде. Если кто-либо изменяет написанный вами код, в какой бы части системы этот код не находился бы, вы должны доверять сделанным изменениям и делать на основе этих изменений выводы. Конечно же, если эти изменения сделаны недостаточно продуманно, на вас лежит ответственность за улучшение текущего положения вещей.

И прежде всего остального вы должны быть готовыми посмотреть в лица своим страхам. Все мы боимся:

- выглядеть глупыми;
- показаться бесполезными;
- стать устаревшими;
- быть недостаточно хорошими.

Без запаса храбрости XP просто не сработает. Вы можете потратить все ваше время, отчаянно пытаясь не совершить ошибку. Вместо этого, если вы желаете, воспользуйтесь помощью команды и взгляните в глаза своим страхам. После этого вы сможете оказаться в деле, вы будете в составе команды с большим удовольствием для себя создавать превосходное программное обеспечение.

Заказчик

Заказчик — это вторая половина базовой двойственности экстремального программирования. Программист знает, как программировать. Заказчик знает, что программировать. Конечно, не с самого начала, но заказчик желает узнать столь же много, сколь знает программист.

Быть заказчиком в XP не так-то просто. Вы должны научиться некоторым важным навыкам, например написанию хороших историй. И ваша целеустремленность в этом деле — это путь к успеху. Однако самое важное — вы должны научиться оказывать на проект мягкое влияние, не будучи в состоянии при этом контролировать его. Силы, которые действуют вне рамок вашего контроля, формируют то, что создается в рамках проекта в такой же степени, как и решения, которые вы принимаете. Изменения в характере функционирования бизнеса, эволюция технологий, состав и возможности команды — все эти факторы оказывают значительное влияние на программное обеспечение, которое разрабатывается в рамках проекта.

Вам придется принимать решения. Для всех тех заказчиков, с которыми мне приходилось работать, это было наиболее сложным навыком. Все они привыкли к информационным технологиям, которые не приносят и половины той пользы, которую им обещают, а та половина, которую получает заказчик, оказывается наполовину неправильной. Заказчики приучены не уступать информационным технологиям ни одной лишней йоты, так как они ожидают, что в любом случае окажутся разочарованными. XP не работает с такими заказчиками. Если вы являетесь заказчиком XP, команда должна быть способной сказать вам с полной уверенностью: «Вот это более важно, чем вот это», «Эта история слишком большая, ее части

будет достаточно», «Этих нескольких историй вполне достаточно». И когда время начинает поджимать, а оно фактически всегда начинает поджимать, команда будет нуждаться в том, чтобы вы заново обдумали свои требования и, возможно, пересмотрели некоторые из них. «Ну что ж, я думаю, что мы вполне сможем обойтись без этого до следующего квартала». Способность принимать подобные решения иногда позволит вам сохранить вашу команду и снизить стресс настолько, что они будут способны сделать для вас все, что в их силах.

Лучшими заказчиками являются те, кто будет на практике использовать разрабатываемую систему. Однако при этом они должны также обладать некоторым перспективным взглядом на проблему, которую предстоит решить. Если вы являетесь одним из таких заказчиков, вы должны следить за тем, чтобы ваши мысли текли в правильном направлении. Иными словами, вы должны думать о том, как обычно выполняются те или иные действия и операции. Если вы на шаг или на два удалены от реального использования системы, вы должны приложить максимальные усилия для того, чтобы максимально точно представлять интересы реальных пользователей системы.

Вы должны научиться писать истории. Поначалу это может показаться почти невыполнимой задачей. Однако в ответ на первые несколько историй команда предоставит вам чрезвычайно полезный набор мнений и рекомендаций. Вы быстро научитесь, насколько емкой должна быть каждая из историй, какая информация должна быть включена в историю, а какой информации в ней быть не должно.

Вы должны научиться писать функциональные тесты. Если вы — заказчик приложения с математической базой, ваша задача упрощается — несколько минут или несколько часов работы с электронной таблицей будет достаточно для того, чтобы сформировать данные для тестового случая. Возможно, команда разработает специально для вас инструмент, упрощающий создание новых тестовых случаев. Программы, основанные на формулировках, также нуждаются в функциональных тестах. Вы должны тесно взаимодействовать с командой для того, чтобы понять, какие вещи полезнее всего протестировать и тесты какого типа являются излишними. Некоторые команды могут даже выделить вам специально техническую помощь для подбора, написания и запуска тестов. Ваша цель: написать тесты, которые позволят вам сказать: «Ну, если все это срабатывает, значит, я уверен в том, что система работает нормально».

Наконец вы должны быть способны продемонстрировать отвагу. Между тем, где вы находитесь сейчас, и тем, куда вы намерены прийти, пролегает долгий путь. Эта команда поможет вам отыскать дорогу, если вы ей поможете в этом.

Тестер

Большая часть обязанностей, связанных с тестированием, лежит на плечах программистов, поэтому роль человека, выполняющего тестирование, в команде XP фокусируется на заказчике. Тестер помогает заказчику в подборе и написании функциональных тестов. Если функциональные тесты не являются частью интеграционного пакета, тестер отвечает за регулярный запуск функциональных тестов и публикацию результатов в обозримом для всех месте.

Тестер XP — это не отдельный человек, специально нацеленный на разрушение системы и унижение программистов. Однако должен быть кто-то, кто будет регулярно запускать тесты (если вы не можете запустить тесты модулей и функциональные тесты одновременно), оповещать команду о результатах тестирования и проверять правильность работы тестирующих инструментов.

Ревизор

Как ревизор вы должны быть совестью команды.

Для того чтобы делать правильные предварительные оценки, вам потребуется обратная связь и практика. Вы обязаны делать множество предварительных оценок и затем следить, насколько реальность соответствует этим оценкам. В следующий раз, когда команда будет формировать предварительную оценку, вы должны быть способны сказать: «Две трети наших предположений в прошлый раз оказались завышенными по крайней мере на 50%». Последующие оценки — это по-прежнему обязанность людей, которые занимаются реализацией того, что оценивается, однако как ревизор вы должны предоставить им обратную связь, указать на неточности с тем, чтобы они смогли сделать оценки лучше, чем в прошлый раз.

Вы также должны следить за общей картиной разработки. На половине работы над очередной итерацией вы должны быть способны сказать команде, удастся ли достигнуть намеченной цели, если следовать ранее определенным курсом, или требуется что-либо изменить. Через две итерации в направлении даты выпуска очередной версии вы должны быть способны сказать команде, сможет ли она выпустить очередную версию без значительных изменений или потребуется чего-либо пересмотреть и переделать.

Как ревизор вы также выполняете функции историка команды. Вы ведете летопись результатов функционального тестирования. Вы вносите записи в журнал обнаруженных дефектов, кто несет ответственность за каждый из дефектов и какие тестовые случаи добавлены для идентификации каждого из дефектов.

Навык, который вы должны развивать в себе больше всех остальных,— это умение собирать необходимую для вас информацию, не беспокоя при этом весь остальной процесс больше, чем это необходимо. Конечно, вы должны немножко побеспокоить работающих соратников для того, чтобы выяснить, сколько времени им в реальности потребовалось для выполнения той или иной задачи. Однако это необходимо делать так, чтобы люди обращали на это как можно меньше внимания. Вы не должны становиться для них занозой в мягком месте так, что они будут избегать встречи с вами.

Инструктор

Как инструктор вы отвечаете за весь процесс разработки. Вы должны обращать внимание на то, что люди отклоняются от заранее обусловленного порядка работы, и вы должны обращать на это внимание всей команды. Вы должны оставаться спокойным в то время, когда все остальные в панике. Вы должны помнить, что за следующие две недели вы сможете сделать объем работ, рассчитанный на две недели и не более (а возможно, и менее) того. Этого либо достаточно, либо нет, третьего не дано.

Каждый в команде XP несет ответственность за осмысление методик XP до определенной степени. Вы же отвечаете за более глубокое их восприятие — какие альтернативные методики могут помочь в решении текущего набора проблем; каким образом XP используется в других командах; какие идеи лежат в основе XP и какое отношение они имеют к текущей ситуации.

Наиболее сложный вывод, который я сделал, исполняя роль инструктора,— это то, что вы работаете лучше, если вы работаете ненапрямую. Если вы видите ошибку в дизайне, прежде всего вы должны решить, является ли это настолько важным, что требуется ваше вмешательство. Каждый раз, когда вы начинаете влиять на поведение команды, вы делаете ее участников менее самостоятельными. Если вы будете руководить ими в большой степени, они потеряют возможность работать без вас. В результате снизится производительность, ухудшится качество и ослабеет моральный дух. Таким образом, прежде всего вы должны определить, стоит ли обнаруженная вами проблема того риска, который вы принимаете, решая вмешаться в обычный ход дела.

Если все же вы приняли решение, что ваше мнение в данной ситуации весомее, чем мнение команды, то вы должны сделать ваше вмешательство как можно менее разрушительным. Например, вместо того, чтобы самостоятельно изменять дизайн системы, будет лучше, если вы предложите такой тестовый случай, удовлетворить который можно, только изменив дизайн.

Это великое искусство — не говорить напрямую, что вы видите, а говорить об этом так, что вся остальная команда тоже начинает видеть это.

Однако в некоторых ситуациях вы должны действовать прямо, прямо до грубости. Уверенные в себе, агрессивные программисты ценны именно тем, что они уверены в себе и агрессивны. Однако при этом они становятся беззащитными перед определенного рода недальновидностью, и единственным лекарством в подобных ситуациях является прямой разговор. Когда вы позволили ситуации ухудшиться до той степени, что нежная рука на загривке не дает желаемых результатов, вы должны быть готовыми схватить удила в обе руки и прикрикнуть: «А ну, п-шо-о-ол!» направляя кого надо в нужную сторону. Но делать это надо только до того момента, пока команда не выедет на нужную дорогу. После этого вы должны вновь ослабить свое влияние.

Здесь я хочу еще кое-что рассказать о роли инструктора. Как инструктор я постоянно обучаю мою команду навыкам XP — простому дизайну, переработке кода, тестированию. Однако я не считаю, что все это должно входить в определение роли инструктора. Если у вас есть команда, которая в достаточной степени технически самодостаточна, однако нуждается в помощи для организации процесса, вы можете быть инструктором и при этом не быть техническим волшебником. В этом случае вы по-прежнему должны добиваться от всех членов команды, чтобы они слушались вас. Однако когда все уже обладают необходимыми навыками, ваша задача будет заключаться в том, чтобы напоминать им о тех методиках, которые они собрались применять в тех или иных ситуациях.

Роль инструктора теряет свою значимость по мере того, как команда «взрослеет» и совершенствуется. В соответствии с принципами распределенного управления и принимаемой ответственности сам процесс XP становится общей ответственностью. На ранних стадиях перехода к XP просить программистов приходится о многих вещах.

Консультант

В рамках проекта XP, как правило, не возникает каких-либо специализаций. Каждый член команды работает в паре с любым из других членов, партнеры в парах часто меняются, пары меняют свой фокус в рамках системы с высокой частотой, и каждый член команды при желании может взять на себя ответственность за выполнение любой задачи. По этим причинам в проекте не возникает черных дыр, связанных с тем, что только один или два человека обладают доскональным знанием о той или иной части системы или обо всей системе в целом.

Это является существенным преимуществом, так как команда становится чрезвычайно гибкой. Однако это является также и слабым местом.

потому что время от времени команда нуждается в глубоких технических знаниях. Благодаря тому что XP концентрируется на простоте дизайна, редко когда в процессе разработки возникает надобность в помощи со стороны специалистов, однако время от времени такое случается.

Когда это происходит, команда нуждается в помощи консультанта. Если вы консультант, значит, вы не привыкли работать в рамках XP. Скорее всего, вы смотрите на то, что делает команда, с некоторым скептицизмом. Однако команда должна обладать четким представлением о том, какую проблему ей требуется решить. Команда должна снабдить вас тестами, которые подскажут вам, когда проблему можно считать решенной (в действительности они будут настаивать на тестах).

Однако при этом они не дадут вам просто уйти и решить проблему самостоятельно. Если команде потребовались глубокие технические знания в некоторой области, значит, скорее всего, это не в последний раз. Теперь задача команды состоит в том, чтобы получить от вас все необходимые знания для того, чтобы решить свою собственную проблему своими силами. Для этой цели один или двое членов команды будут сидеть рядом с вами, когда вы будете решать проблему. Скорее всего, они будут задавать вам множество вопросов. Они будут внимательно оценивать предлагаемый вами дизайн и сделанные вами предположения, при этом они будут стараться понять, нельзя ли сделать что-либо более простое и при этом добиться решения проблемы.

И когда вы наконец решите поставленную перед вами задачу, они, скорее всего, выбросят все, что вы сделали, и попытаются заново решить проблему своими собственными силами. Не переживайте и не расстраивайтесь. В определенной мере они занимаются этим каждый день и, возможно, раз в месяц им приходится выкидывать на свалку результаты дневной работы.

Большой босс

Если вы — большой начальник, значит, команда больше всего желает видеть в вас **храбрость,** уверенность и время от времени возникающее желание убедиться в том, что команда делает именно то, что она обещала сделать. Поначалу работать с командой для вас будет совсем несложно. Они будут предлагать вам проверять их почаще. В любой ситуации они **будут** объяснять вам последствия любых изменений. Например, если вы не выделите им нового тестера, о котором они просят, они вам доступно **объяснят,** как именно и почему сместится график работ и насколько отдалится дата выпуска очередной версии. Если вам не понравится их ответ, они предложат вам уменьшить объем работ по проекту.

Команда XP стремится к откровенной и правдивой коммуникации. Они не хнычут и не плачутся. Они просто хотят, чтобы вы как можно раньше узнали о том, что реальность начинает отличаться от заранее сформированного плана, благодаря чему у вас будет больше времени, чтобы среагировать должным образом.

Команда желает, чтобы у вас было достаточно храбрости, потому что то, чем они занимаются, иногда выглядит для вас полным безумством, особенно если ранее вы уже работали в области разработки программного обеспечения. Некоторые идеи покажутся вам вполне резонными и оправданными, например ориентация на постоянное и надежное тестирование. Другие методики поначалу покажутся бессмысленными. Например, вам покажутся подозрительными утверждения о том, что программирование в парах является более эффективным методом разработки кода и что постоянный пересмотр и переделка дизайна — это менее рискованный метод проектирования программных систем. Но посмотрите, что выходит из рук этих ребят. Если это не срабатывает, вы можете вмешаться. Однако если у них получается, значит вы в выигрыше, так как у вас есть команда, которая работает с высокой продуктивностью, которая делает заказчиков счастливыми и которая делает все от нее зависящее для того, чтобы у вас никогда не было неприятных сюрпризов.

Это не означает, что у команды никогда не будет сбоев. Время от времени затруднительные ситуации, конечно же, будут возникать. Но вы будете смотреть на то, что у них получается, и для вас не будет иметь значения, что именно они делают для того, чтобы добиться результата. Вы просите у них объяснить вам это, и само по себе объяснение тоже не будет иметь значения, так как именно в такие моменты команда благодаря вам получает возможность остановиться и взглянуть на свою деятельность. Вы находитесь на своем месте не просто так. Команда использует это для своих собственных целей тогда, когда ей это нужно. А когда ей это не нужно, откровенно говоря, команда обходит это.

Правило 20 на 80

23

Полная отдача от XP получается только тогда, когда в силу вступают все методики. Многие практики можно вводить в силу постепенно, однако если все они введены в **действие,** общий эффект от их использования равен произведению между ними.

Разработчики программного обеспечения привыкли иметь дело с правилом 20 на 80, которое утверждает, что 80% пользы исходит из 20% работы.

Чтобы правило 20 на 80 работало, рассматриваемая система должна обладать элементами управления, в достаточной степени независимыми друг от друга. Например, когда я занимаюсь оптимизацией производительности программы, я могу вносить изменения в несколько разных мест, при этом желательно, чтобы при внесении изменения в одно место это изменение не оказывало бы существенного влияния на все остальные места, в которых я могу повлиять на систему. Я никогда не должен попадать в ситуацию, в которой при изменении одного элемента системы, оказывающего наибольшее влияние на время исполнения программы, я обнаруживаю, что после внесения этого изменения я уже не могу воздействовать на второй по важности элемент системы.

Вард Каннингхэм (Ward Cunningham) рассказывал мне о книге, посвященной освоению техники горнолыжного спуска. Эта книга называлась *The Athletic Skier* «Лыжник-спортсмен»[1]. Половина книги была посвящена настройке и подгонке горнолыжных ботинок таким образом, чтобы вы уверенно стояли на лыжах, чувствовали склон и поддерживали равновесие во время спуска. После этого в книге говорилось: «Однако после того, как вы выполните 80% всех этих упражнений, вы улучшите вашу ситуа-

[1] Warren Witherell and Doug Evrard, *The Athletic Skier,* Johnson Books, 1993.

цию лишь на 20%». Далее объяснялось, что существует огромная разница между состоянием равновесия и состоянием потери равновесия. Если вы немножко не в равновесии, это может означать, что вы полностью потеряли равновесие. На это влияет огромное количество факторов, например качество настройки ваших горнолыжных ботинок. Если хотя бы один из них настроен неправильно, значит, вы потеряете баланс. По мере того как вы учитываете в своей подготовке все большее количество факторов, ситуация будет очень медленно улучшаться. Однако когда вы внесете в свою подготовку несколько самых последних изменений, вы получите существенные, кардинальные улучшения вашей техники спуска.

Я думаю (и это просто моя гипотеза), что подобное положение вещей имеет место и в отношении XP. Методики и принципы XP работают вместе друг с другом, и в результате получается сила, которая действует сильнее, чем сумма составляющих ее частей. Вы не просто выполняете тестирование, вы тестируете систему, которая должна быть простой, и при этом система становится простой благодаря тому, что вы программируете в паре и ваш партнер следит за тем, чтобы вы должным образом перерабатывали код, напоминает вам о том, что вы должны писать больше тестов, и одобрительно хлопает вас по плечу, когда вы избавляетесь от излишней сложности кода, и...

Тут возникает дилемма. Не является ли XP вещью из разряда «все или ничего»? Должны ли вы обязательно следовать каждой из входящих в XP методик или вы рискуете не заметить вообще никаких улучшений? Вовсе нет. Вы можете получить заметное увеличение продуктивности, даже если вы будете использовать только часть методик XP. Однако я подозреваю, что вы сможете получить гораздо больше, если вы будете по максимуму использовать все методики в комплексе.

Что делает XP сложной?

24

Несмотря на то что отдельные методики без труда могут исполняться обычными программистами в индивидуальном порядке, соединение всех кусков воедино и поддержание их в этом состоянии — далеко не простая задача. Сложной XP становится в основном из-за эмоций, в особенности из-за страха.

Когда люди слушают то, что я им рассказываю об XP, они говорят: «То, о чем ты рассказываешь, кажется таким простым!» Да, это действительно очень просто. Не надо обладать ученой степенью в области компьютерных наук для того, чтобы участвовать в XP-проекте (в действительности ученая степень частенько является одним из серьезных мешающих факторов).

XP очень проста в своих деталях, однако ее сложно реализовать на практике.

Повторю это снова. XP легко понять, но сложно реализовать. Именно так. Методики, составляющие XP, может изучить каждый, кто занимается программированием. Это несложная часть работы. Сложная часть — это собрать и поддерживать в состоянии баланса. Отдельные составные части XP поддерживают друг друга, однако существует множество проблем, опасений, страхов, событий и ошибок, которые могут вывести процесс из состояния баланса. Причина, по которой вам приходится «жертвовать» наиболее технически опытным членом команды, чтобы он исполнял роль инструктора, состоит в том, что задача поддержки процесса в состоянии баланса — это очень непростая задача.

Я не хочу пугать вас. Я не хочу пугать вас больше, чем это необходимо. XP может с успехом применяться большинством команд, занимающихся

разработкой программного обеспечения. (Исключения рассматриваются в следующей главе.)

Я хочу рассказать вам о некоторых моментах, показавшихся мне сложными, когда я применял XP в отношении моего собственного кода, равно как и когда я инструктировал команды, решившие внедрить у себя XP. Я не хочу нагонять на вас излишний страх, но когда переход к XP будет для вас сложным (а я вам обещаю, что такие моменты будут), вы должны знать, что вы не одиноки. Сложности у вас возникают потому, что то, чем вы занимаетесь, — это сложно.

Сложно делать вещи простыми. Это звучит неправдоподобно, но иногда проще сделать что-нибудь более сложное, чем сделать что-нибудь простое. Это особенно верно, если в прошлом вы долго и успешно делали сложные вещи. Умение видеть мир в самых простых терминах — это очень сложное умение. Проблема состоит в том, что вы должны изменить свою систему ценностей. Вместо того чтобы восхищаться тем, как кто-то (например, вы) добивается функционирования чего-либо сложного, вы должны научиться относиться к сложности с презрением и не успокаиваться до тех пор, пока вы не добьетесь тех же самых результатов от более простой системы.

Сложно признавать то, что вы чего-то не знаете. XP — это дисциплина, которая основана на предположении, что вы можете работать не быстрее, чем вы можете получать важную информацию, то есть обучаться, поэтому освоение XP может быть для вас персональным испытанием. А если вы обучаетесь, это означает, что до этого вы чего-то не знали. Сможете ли вы без каких-либо опасений и стеснений прийти к заказчику и попросить его объяснить вам то, что для него является простейшей, элементарнейшей концепцией? Сможете ли вы без опасений и стеснений обратиться к вашему партнеру по паре и сказать ему, что существуют некоторые тривиальные, базовые сведения об информационных технологиях, которые вы, откровенно говоря, недостаточно хорошо изучили в школе. Или, может быть, просто забыли.

Сложно взаимодействовать с соратниками. Вся наша система образования ориентирована на индивидуальное продвижение вперед. Если вы работаете над проектом не один, учитель называет это жульничеством, подсказкой и нечестностью и наказывает вас. В большинстве компаний система вознаграждений основана на индивидуальных оценках и повышениях (часто такую систему называют игрой с нулевой суммой), а это тоже стимулирует индивидуальное мышление. В рамках XP вы должны оттачивать свое мастерство благодаря помощи ваших сотоварищей. Дисциплина XP основана на тесном взаимодействии всех членов команды.

Сложно разрушать эмоциональные стены. Гладкое течение проекта XP основано на гладком выражении эмоций. Если кто-либо чувствует себя расстроенным или озлобленным и никому не говорит об этом, через некоторое время это сказывается на производительности команды. Мы привыкли отделять нашу эмоциональную жизнь от нашей деловой жизни, однако команда не может функционировать эффективно, если не будет происходить живого общения, если не будут подчеркиваться страхи, если не будет разряжаться гнев, если счастье не будет делиться между всеми членами команды.

В свое время я пытался разрабатывать программное обеспечение так, как будто у меня нет никаких эмоций, при этом я держался на некотором расстоянии от своих соратников. Это не было эффективным. Когда я говорю другим о том, что чувствую, и слушаю остальных, когда они говорят, что они чувствуют, весь процесс протекает более гладко.

Методики XP настолько далеки от того, о чем мы говорили и о чем мы слышали и с чем мы добивались успеха в прошлом, что вся методика XP выглядит странно для тех, кто с ней раньше не сталкивался. Наибольшую сложность представляет противоречивость XP. Когда я встречаюсь с очередным новым менеджером, я часто опасаюсь, что он воспримет то, о чем я говорю, как нечто радикальное, или безумное, или непрактичное. Однако я не знаю другого, лучшего способа разработки программного обеспечения, поэтому со временем я уже научился преодолевать в себе этот страх. Однако когда вы приступаете к объяснению сути XP незнакомому с этой дисциплиной человеку, вы должны быть готовы к тому, что он отнесется к XP достаточно жестко.

Небольшие проблемы могут привести к значительным эффектам. Проверки и балансирование XP достаточно надежны, однако процесс может в некоторой степени варьироваться. При этом необходимо учитывать, что даже на первый взгляд незначительные вещи могут привести к значительным отклонениям. Когда мы работали над системой управления выплатами в команде Chrysler C3, у команды возникли проблемы с реализацией работы к сроку. Мы никак не могли добиться того, чтобы наши предварительные оценки совпадали с реальностью. В каждой очередной итерации мы каждый раз не успевали реализовывать одну или две истории. Для того чтобы определить корень проблемы, мне потребовалось три или четыре месяца. Я услышал, как кто-то говорит о «синдроме первого вторника». Я переспросил, что это означает, и один из членов команды объяснил мне: «Синдром первого вторника — это ощущение, которое возникает на следующий день после совещания, посвященного планированию очередной итерации, когда ты приходишь на работу, смотришь на свои исто-

рии и не представляешь себе, как можно реализовать все это за то время, которое было оценено как достаточное».

Дело оказалось в следующем. Изначально я описал процесс следующим образом.

1. Распределение заданий между участниками.
2. Оценка заданий участниками в индивидуальном порядке.
3. Перераспределение в случае, если кто-то оказывается перегруженным.

Однако команда захотела избежать третьего шага, в результате было предложено изменить процесс следующим образом.

1. Коллективная оценка заданий.
2. Распределение заданий между участниками.

Проблема заключалась в том, что связанная с задачей оценка не принадлежала человеку, принимающему на себя ответственность за выполнение этой задачи. Иными словами, получалось, что один человек (или команда в целом) решает, за какое время можно справиться с задачей, а другой человек вынужден работать над решением этой задачи, пытаясь успеть в заданный ему срок. В результате на следующий день исполнитель приходит на работу, смотрит на задачу и с ужасом думает: «Разве можно справиться с этим за три дня? Я даже не владею всеми необходимыми для этого знаниями». Согласитесь, что это не самое продуктивное состояние для программиста. Таким образом, на каждую итерацию борьбы с синдромом первого вторника у каждого члена команды уходил день или два. Нет ничего удивительного в том, что они не могли достичь всех поставленных перед ними целей.

Я рассказал эту историю для того, чтобы проиллюстрировать, что небольшие проблемы во время проекта могут стать причиной значительных эффектов. Я не имел в виду, что вы обязаны делать все именно так, как описано в данной книге. Вы можете попробовать сформировать свой собственный процесс разработки. Но именно это и делает XP сложной — если вы принимаете на себя ответственность за формирование собственного процесса разработки, значит, вы принимаете на себя ответственность за наблюдение за возникновением и решением разного рода проблем.

Управление проектами, основанное на незначительных отклонениях то в одну сторону, то в другую сторону, идет в разрез с традиционными методиками управления, основанными на предварительном планировании

с последующим следованием жестко заданному плану. Именно такая дисциплина применяется в большинстве организаций. Финальная сложность, благодаря которой XP-проект вполне может умереть, состоит в том, что управление проектом в соответствии с метафорой управления автомобилем совершенно неприемлема в рамках культуры производства, принятой во многих компаниях. Раннее предупреждение о проблемах рассматривается как признак слабости или как жалобы на жизнь. Если ваша компания потребует от вас действовать вопреки принципам, которые вы избрали для себя определяющими, вам потребуется храбрость.

Когда не следует использовать XP

25

Точные пределы использования XP еще не до конца исследованы. Однако есть известный набор факторов, который делает применение XP невозможным, — слишком большие команды, недоверчивые **заказчики,** технология, которая не позволяет легко и просто вносить изменения.

В рамках XP используются методики, которые сами по себе являются неплохими вне зависимости от того, что вы думаете об общей картине. Вы должны применять их. Точка. Тестирование — хороший пример. Игра в планирование тоже, скорее всего, будет работать, несмотря на то что вы будете большее время тратить на формирование оценок и предварительное проектирование. Однако, как предписывает правило 20 на 80, существует значительная разница между применением всех методик и применением только некоторых из них.

Дисциплину XP не следует применять абсолютно для любых проектов. При некоторых комбинациях времени, места, команды разработчиков и заказчиков проект XP может лопнуть, как воздушный шарик. Очень важно не использовать XP для таких проектов. XP необходимо использовать там, где эта дисциплина обеспечивает реальные преимущества, и ее не нужно использовать там, где она может дать сбой. Именно этому и посвящена данная книга — прочитав ее, вы сможете решить, когда можно использовать XP, а когда этого лучше не делать.

Я не буду говорить вам нечто вроде: «Не следует использовать XP для разработки программ управления стратегическими ракетами». Я никогда не разрабатывал программное обеспечение подобного рода, поэтому я не могу сказать вам, на что это похоже. Исходя из этого я не могу сказать вам, будет ли работать XP в данной ситуации. Однако при этом я так же не могу с уверенностью сказать вам, что в данном случае XP абсолютно

точно не будет работать. Если вы занимаетесь разработкой программ для стратегических ракет, вы должны самостоятельно решить, подходит ли для вас XP или нет.

Все же я достаточно часто сталкивался с неудачами XP для того, чтобы рассказать вам о некоторых факторах, благодаря которым XP совершенно точно не будет работать. Воспринимайте все, о чем я буду рассказывать, как перечень рабочих сред, о которых я точно знаю, что в них XP работает не самым лучшим образом.

Самым большим барьером, стоящим перед XP, является культура. Но не национальная культура, которая безусловно тоже немаловажна, а бизнес-культура. Любой бизнес, который привык вначале разрабатывать план, а затем действовать в строгом соответствии с этим планом, будет сильно конфликтовать с командой, которая намерена постоянно менять план своих действий по мере работы над проектом.

Любой план предусматривает использование емкой, достаточно большой и тщательно продуманной спецификации. Если заказчик или менеджер настаивает на составлении полной спецификации или подробного анализа или разработке полноценного дизайна перед тем, как можно будет сесть за написание кода, тогда возникает сильное трение между культурой команды и культурой заказчика или менеджера. В рамках проекта все же можно будет использовать XP, однако это будет непросто. Вы предлагаете заказчику или менеджеру способ работы над проектом в форме диалога (игра в планирование), который подразумевает, что они будут находиться в постоянной связи с проектом. Для человека, который и без того очень сильно занят, это может оказаться неприемлемым.

Но однажды я работал для одного банка, представители которого просто очень любили большие куски бумаги. В процессе работы над системой они постоянно настаивали на том, чтобы мы документировали систему. Мы постоянно отвечали им, что если они хотят получить меньше функциональности и больше бумаги, мы будем рады удовлетворить их просьбу. Мы слушали их просьбы о документации в течение нескольких месяцев. По мере того как проект продвигался вперед и становилось ясным, что тесты не только обеспечивают стабильность системы, но и отлично описывают ее поведение, требования о документировании становились все тише и тише. Все же они продолжали настаивать на своем. В самом конце работы над проектом менеджер по разработке сказал нам, что все, что ему нужно, это четырехстраничное обозрение основных объектов системы. Он пришел к выводу, что любой человек, который не в состоянии получить необходимую ему информацию из кода и тестов, вообще не должен трогать код.

Еще одна культура, в рамках которой не следует применять XP, это культура, в рамках которой вы должны работать внеурочное время для того, чтобы доказать свою преданность компании. Вы не можете работать в стиле XP, если вы устали. Если объем работы, выполняемый командой, работающей с максимальной скоростью, недостаточен для компании, значит, XP — это не ваше решение. Если вы работаете над проектом XP сверхурочно в течение двух недель подряд — это верный признак того, что вы делаете что-то не так. Вместо того чтобы продолжать работать в напряженном режиме, лучше попытаться обнаружить проблему и устранить ее.

Иногда очень умные программисты с трудом овладевают XP. Для очень умных людей тяжело поменять их умение делать правильные дальновидные предположения на тесную коммуникацию и постоянную эволюцию системы.

Огромное значение имеет масштаб проекта. Скорее всего, вам не удастся эффективно использовать XP, если над проектом работает сотня программистов. Пятьдесят программистов — это тоже слишком много. Скорее всего, и двадцать программистов будет много. Десять программистов — это определенно подходящее количество. Если в команде всего три или четыре программиста, вы можете безбоязненно отбросить некоторые из методик, сфокусированных на координации, например игру в планирование итерации. Объем функциональности, который требуется реализовать, и количество людей, которые работают над реализацией этой функциональности, связаны зависимостью, которая ни в коем *случае* не является линейной. Если у вас крупномасштабный проект, вы можете экспериментировать с XP — попробуйте использовать ее в течение месяца с небольшой командой, чтобы проверить, как быстро будет продвигаться разработка.

Наиболее узким местом при использовании XP в крупных проектах является однопоточный интеграционный процесс. Вы должны как-либо расширить этот процесс, чтобы обеспечить интеграцию нескольких источников кода на одном компьютере.

Вы не можете использовать XP в случае, если имеете дело с технологией, которая подразумевает экспоненциальный рост затрат, связанных с внесением в систему изменений. Например, если вы имеете дело с мэйнфреймом, планируете использовать установленную на нем реляционную базу данных и не уверены в том, что схема реляционной базы данных в точности соответствует тому, что вам нужно. В подобной ситуации вы не должны использовать XP. XP основывается на чистом и простом коде. Если вы усложняете код для того, чтобы избежать модификации 200 существующих приложений, в самом скором времени вы потеряете гибкость, ради которой вы, собственно, и решили использовать XP.

Еще одним технологическим барьером, препятствующим использованию XP, является рабочая среда, в которой для обратной связи требуется слишком длительное время. Например, если для компиляции и компоновки вашей системы требуется 24 часа, вы вряд ли сможете интегрировать, собирать и тестировать по несколько раз в день. Если перед тем, как приступить к использованию системы на производстве, вы вынуждены пройти двухмесячную процедуру проверки качества, вы не сможете быстро получать сведения о том, как ведет себя система в условиях реального производства при добавлении или изменении той или иной функциональности.

Я работал в средах, в которых было невозможно тестировать программное обеспечение в обычном смысле этого слова, — в условиях производства система работает на компьютере стоимостью в один миллион долларов, который загружен до предела, и у вас нет еще одного лишнего миллиона долларов. Или существует такое большое количество комбинаций возможных проблем, что вы просто не можете выполнить маломальски осмысленный набор тестов за время, меньшее, чем один день. В подобной ситуации самым правильным решением будет заменить тестирование длительным предварительным размышлением. Однако в этом случае то, что вы делаете, нельзя будет назвать XP. Когда я программирую в рабочей среде подобной категории, я не чувствую себя способным свободно видоизменять дизайн системы; я обязан сформировать необходимую гибкость заранее. В этом случае вы по-прежнему можете производить качественное программное обеспечение, однако при этом вы не должны использовать XP.

Помните историю о программистах, офисы которых размещались в разных концах здания? Если вы работаете в неподходящих физических условиях, XP не будет работать. Как было отмечено, для целей XP я использовал большую общую комнату с мощными компьютерами в центре и с небольшими отделениями вдоль стен. Однако это только один из возможных вариантов. Вард Каннингхэм (Ward Cunningham) рассказывал мне о проекте WyCach, при работе над которым каждому программисту выделили по отдельному небольшому рабочему месту, отделенному от внешнего мира перегородками из ДСП. Однако каждый такой мини-офис был достаточно просторным для того, чтобы вместить двух человек. Если два человека хотели работать в паре, они выбирали один из офисов. Если вы абсолютно точно не можете передвигать столы, если уровень шума высок настолько, что сильно мешает разговору, если вы не можете расположиться достаточно близко для того, чтобы обеспечить хорошую коммуникацию, вы не сможете использовать XP с максимальной эффективностью.

Что не сработает абсолютно точно? Если ваши программисты размещаются на двух этажах, вы можете забыть об XP. Если программисты расположены на одном этаже, но далеко друг от друга, вы тоже можете забыть об XP. Если программисты географически удалены друг от друга, то, вы можете попытаться использовать XP при условии, что над разными логически связанными частями проекта будут работать две команды и что они смогут работать в условиях ограниченной коммуникации между собой. Можно начать работу над проектом при помощи одной команды, выпустить первую версию, затем разделить команду согласно географическим границам, поручить каждой из команд реализацию одной из логически отдельных частей приложения и развивать каждую из частей по отдельности.

Наконец, вы абсолютно точно не сможете работать в стиле XP, если в одной комнате с вами кричит ребенок. В этом вы мне можете абсолютно точно довериться.

XP в работе

26

XP может применяться при любой форме контракта, однако разные формы предусматривают использование разных вариаций XP. В частности, контракты с фиксированной ценой и фиксированным объемом работ благодаря игре в планирование становятся контрактами с фиксированной ценой, фиксированной датой и приблизительно фиксированным объемом работ.

Как XP вписывается в общераспространенные разновидности бизнес-практики? Неправильная форма контракта может легко разрушить проект вне зависимости от инструментов, технологии и таланта.

В данной главе анализируются некоторые бизнес-аспекты разработки программного обеспечения, а также то, как вы можете использовать их совместно с XP.

Фиксированная цена

Практика показывает, что наибольшие проблемы с использованием XP возникают в случае, если работа идет над контрактом с фиксированной ценой. Можно ли играть в игру в планирование, если вы имеете дело с контрактом фиксированной цены/фиксированной даты/фиксированного объема работ? В результате получается контракт фиксированной цены/фиксированной даты/несколько варьируемого объема работ.

Каждый проект с фиксированной ценой и фиксированным объемом работ, с которым мне приходилось иметь дело, заканчивался тем, что обе стороны говорили друг другу: «Требования неясны». В результате работы над проектом с фиксированной ценой и фиксированным объемом работ две стороны начинают двигаться в противоположных направлениях. Поставщик продукта желает сделать как можно меньше, а заказчик желает получить как можно больше. В условиях подобного трения обе стороны

желают успешного выполнения проекта, поэтому они отказываются от своих изначальных целей, но трение остается.

В рамках XP взаимоотношения меняются малозаметным, но важным образом. Изначально разговор об объеме работ обсуждается с использованием слова «например». «Например, за 5 000 000 немецких марок мы могли бы реализовать все эти истории за 12 месяцев». Заказчик должен решить, стоят ли эти истории пяти миллионов немецких марок. Если эти истории — это все, что должна реализовать команда, то это хорошо. Существует вероятность, что заказчик заменит некоторые из историй более полезными для него. Никто не станет жаловаться, если вместо чего-либо он получит что-либо другое, более ценное. Однако недовольство и жалобы возникают тогда, когда кто-либо получает что-либо, о чем он просил, но при этом оказалось, что это не то, что ему нужно.

Вместо фиксированных цены/даты/объема работ команда XP предлагает нечто наподобие подписки. Команда обязуется работать на заказчика с максимальной скоростью в течение определенного промежутка времени. Они будут следить за мнением заказчика относительно направления развития разработки. В начале каждой итерации заказчик получает формальную возможность изменить направление развития разработки и добавить совершенно новые истории.

Еще одно различие, связанное с применением XP, заключается в использовании небольших версий. Вы никогда не должны работать над проектом XP в течение 18 или даже 12 месяцев, не внедрив его в производственных условиях. Когда команда заключает контракт на реализацию историй в течение 12 месяцев, они играют с заказчиком в игру в планирование для того, чтобы определить объем работ, связанных с первой версией. Таким образом, когда речь идет о 12-месячном контракте, система начнет эксплуатироваться в производственных условиях спустя три или четыре месяца после начала работ, после этого новые версии будут выпускаться с периодичностью в один или два месяца. Постепенное введение системы в эксплуатацию обеспечивает заказчику возможность расторгнуть контракт в случае, если процесс разработки идет медленнее, чем планировалось, или если в результате изменения бизнес-условий реализация всего проекта потеряла смысл. Кроме того, так как проект развивается от итерации к итерации, на шкале времени у заказчика появляются точки, в которых он получает возможность изменить направление развития проекта.

Разработка чужими силами

Когда разработка осуществляется руками сторонних исполнителей (outsourcing), в результате у заказчика оказывается кусок кода, который не-

известно как поддерживать и модифицировать. В этом случае есть три варианта:

- новую эволюцию системы можно выполнить своими силами;
- можно нанять для выполнения модификации системы тех же самых разработчиков, которые работали над ее созданием (однако они могут попросить за это слишком много денег);
- можно обратиться к другим разработчикам, которые плохо знакомы с кодом.

Если вы этого хотите, вы можете сделать это с использованием XP. В этом случае либо команда идет работать к заказчику, либо заказчик посылает своих представителей к разработчикам. Они играют в игру в планирование для того, чтобы решить, что делать. Когда контракт завершается, команда уходит и оставляет заказчику код.

В некотором роде это может быть лучше, чем общепринятая форма соглашения о выполнении работ на стороне. У заказчика есть набор функциональных тестов и тестов модулей, которые позволяют ему удостовериться в том, что любые внесенные в систему изменения не нарушают существующей функциональности. Для заказчика будет удобно, если какой-либо его представитель будет стоять за спиной у программистов и вникать в то, как система устроена внутри. При этом заказчик получит возможность более точно руководить разработкой.

Разработка своими силами

Наверное, вам показалось, что я не в восторге от выполнения разработки чужими руками. Дело в том, что если разработка выполняется на стороне, это, как правило, означает, что существует некоторый акт приема-сдачи работы. Проект передается заказчику в виде большого единого продукта, готового к употреблению. Это нарушает принцип постепенного изменения. Однако существует некоторая весьма удобная вариация на эту тему, которую можно реализовать благодаря использованию XP. Что, если вы постепенно будете заменять членов команды, занимающейся разработкой, техническими специалистами, которые являются сотрудниками фирмы-заказчика? Именно это я и называю «выполнением разработки своими силами» (insourcing).

Разработка своими силами — это подчас малоэффективное занятие, так как собственные технические сотрудники подчас не обладают необходимыми для этого знаниями и опытом. Если разработка выполняется силами сторонних специалистов, заказчик получает в свое распоряжение систему, разработанную с использованием опыта и навыков специалистов

действительно высокого класса, однако при этом он не знает, как устроена система и что делать, если возникает надобность в ее модификации. Однако если в процессе работы вы постепенно заменяете чужих разработчиков своими, вы со временем постепенно перекладываете ответственность за разработку с чужих плеч на свои плечи. В результате у заказчика появляются знания об устройстве и функционировании системы, благодаря чему снижается риск того, что заказчик получит программу, которую он не сможет поддерживать.

Давайте рассмотрим пример простого соглашения между некоторым заказчиком и командой из десяти (10) разработчиков. Контракт заключается на 12 месяцев. Изначальная разработка осуществляется в течение 3 месяцев. Затем первая версия продукта начинает эксплуатироваться на производстве. После этого каждая очередная версия выпускается с периодичностью раз в месяц в течение 9 месяцев. На время изначальной разработки заказчик вводит в состав команды разработчиков одного своего представителя, который является техническим специалистом. После этого каждый очередной месяц заказчик вводит в состав команды по одному новому своему техническому специалисту, а исполнитель выводит из состава команды разработчиков одного из своих людей. В конце работы над заказом половина команды разработчиков — это люди заказчика, которые готовы осуществлять поддержку кода программы и продолжать дальнейшую разработку, правда, с меньшей скоростью.

В отличие от ситуации, когда разработка целиком и полностью возлагается на сторонних разработчиков, в условиях, когда состав команды разработчиков изменяется, разработка не может выполняться с такой же скоростью, как если бы состав команды оставался бы неизменным. Однако получаемое при этом снижение риска, возможно, стоит того.

XP обеспечивает нормальное исполнение такой схемы за счет того, что команда постоянно измеряет скорость, с которой она работает. По мере того как происходит замена членов команды, скорость работы команды может меняться как в худшую, так и в лучшую сторону. Измеряя получаемую производительность, команда может вносить изменения в план итераций и влиять на объем работ в ходе игры в планирование. По мере того как эксперты будут уходить, а на их место будут приходить менее опытные люди, команда может выполнять переоценку оставшихся историй.

Время и материалы

В контрактах категории «время и материалы» (Time and Materials, T&M) команда XP получает почасовую оплату. Все остальное работает так, как описывалось ранее.

Проблема контрактов T&M состоит в том, что мотивы исполнителя идут в разрез с мотивами заказчика. Исполнитель желает в течение как можно более длительного времени задействовать в работе над проектом как можно больше людей для того, чтобы максимизировать прибыль. Кроме того, исполнитель имеет тенденцию принуждать разработчиков работать сверхурочно, чтобы увеличить ежемесячную прибыль. Заказчик желает реализовать как можно больший объем функциональности за как можно более короткий срок с использованием как можно меньшего количества людей.

Благодаря хорошим отношениям между заказчиком и исполнителем схема T&M может сработать, однако трение между ними всегда будет существовать.

Премия за завершение

Отличный способ уладить разногласия между заказчиком и исполнителем в рамках контракта с фиксированной ценой или T&M — это учреждение премии за завершение проекта в срок. В некотором смысле премия за завершение проекта к сроку — это легкая добыча для команды XP. Игра в планирование предоставляет команде столь удобный контроль над проектом, что, скорее всего, она без труда сможет получить эту премию.

Обратной стороной премии за завершение работы в срок является штраф за опоздание. Если разработчики не успевают доделать проект к назначенной дате, заказчик платит им меньше, чем договаривались. И вновь благодаря игре в планирование команда XP получает преимущество. Команда всегда уверена в том, что завершит работу в срок, поэтому ей вряд ли придется терять деньги.

Когда премия за завершение в срок или штраф за опоздание используются в рамках XP, необходимо обратить внимание на важную особенность: игра в планирование подразумевает, что объем работ, связанных с проектом, неизбежно будет изменяться по мере работы над проектом. В процессе работы неизбежно будет возникать необходимость добавить новые истории и убрать из технического задания некоторые старые истории. Если заказчик горит желанием взять исполнителя «за жабры», он может сказать примерно следующее: «Сегодня первое марта, а полный набор историй, указанных в изначальном контракте, все еще не реализован. Никаких премий! Теперь вы будете платить неустойку!» Заказчик может сказать подобное даже в случае, если система уже успешно эксплуатируется на производстве.

Как правило, подобной проблемы не возникает. Если наступает Рождество, а под елкой уже лежат подарки, у заказчика вряд ли возникает

желание сверять точное соответствие этих подарков со списком, который он указал в письме к Санта-Клаусу, особенно если сам заказчик просил о тех или иных заменах.

Если вы опасаетесь, что заказчик будет придираться, указывая вам на изначально составленный им список историй с целью отказать вам в премии, лучше вообще не иметь дело с таким заказчиком. Я не рекомендую вам подписывать с ним какие-либо контракты.

Раннее закрытие проекта

Одним из преимуществ XP является то, что заказчик получает возможность видеть, что именно выходит из рук исполнителя по мере работы над проектом. Что, если на полдороге заказчик придет к выводу, что весь проект потерял для него актуальность? В этом случае, очевидно, для заказчика будет удобнее отказаться от дальнейшей разработки. На этот случай в контракте следует предусмотреть специальный параграф, который позволяет заказчику остановить работу над проектом и выплатить часть общей стоимости проекта пропорционально сделанной работе. Возможно, следует предусмотреть дополнительную выплату для компенсации исполнителю, так как в подобной ситуации исполнитель будет вынужден некоторое время тратить на поиск нового контракта.

Программные инфраструктуры

Программная инфраструктура (framework) — это некоторый универсальный исходный код, который может использоваться в качестве основы при разработке разнообразных приложений определенной категории. В программную инфраструктуру могут входить библиотека классов, шаблоны программ и т. п. Программная инфраструктура может быть коммерческим продуктом, но это может быть и продукт для внутреннего пользования.

Можно ли использовать XP для разработки программных инфраструктур? Одно из правил XP гласит, что вы должны удалять из системы любую функциональность, которая в данный момент не используется, если следовать этому правилу, то вы должны удалить всю инфраструктуру

XP не очень хорошо приспособлена для разработки кода, использование которого планируется не сейчас, а в дальнейшем. В рамках XP-проекта вы не можете потратить шесть месяцев на разработку инфраструктуры, а затем приступить к ее использованию. Подход, подразумевающий разделение команды на группу разработки инфраструктуры и группу разработки приложения, тоже не подходит. В XP мы занимаемся разработкой конкретных приложений. Если после нескольких лет постоянных переработок дизайна начинают вырисовываться некоторые универсальные

абстракции общего использования, значит, настало время подумать о том, как сделать их более широко доступными.

Если цель проекта — разработка инфраструктуры для внешнего использования, вы по-прежнему можете использовать XP. В игре в планирование на стороне бизнеса играет программист, который занимается разработкой приложений на базе инфраструктуры, которую вы разрабатываете. Возможности инфраструктуры превращаются в истории.

После того как инфраструктура начинает использоваться вне команды, вы можете применять более консервативный подход к переработке кода. Вы предупреждаете заказчиков о предстоящем изменении или отказе от использования той или иной возможности и продолжаете эволюцию интерфейса вашей инфраструктуры, в течение некоторого времени сохраняя в ней как старую, так и новую версию интерфейса.

Продукты широкого использования

XP можно использовать также для разработки программных продуктов для широкого потребительского рынка. В этом случае роль бизнеса в игре в планирование играет отдел маркетинга. Его сотрудники идентифицируют, в каких историях нуждается рынок, какой объем каждой из историй необходимо реализовать и в каком порядке следует реализовать эти истории.

Иногда бывает полезно включить в состав игроков, играющих на стороне бизнеса, пользователя-эксперта. Например, компании, занимающиеся разработками компьютерных игр, могут нанять в качестве экспертов заядлых игроков в компьютерные игры, которые будут тестировать производимые ими игры. Производители финансовых программ могут нанять в качестве пользователей-экспертов профессиональных финансистов. Если вы производите музыкальный редактор, вы можете пригласить к себе в качестве эксперта профессионального композитора или музыканта. Пользователь-эксперт — это именно тот человек, который будет определять, какую из историй следует в первую очередь реализовать в следующей версии вашего программного продукта.

Заключение 27

Все методики основаны на страхе. Вы пытаетесь развить у себя привычки, которые помогут вам не допустить, чтобы ваши страхи воплотились в реальность. В этом отношении XP ничем не отличается от любой другой методики. Разница состоит в том, что страхи запечатлены в XP. Методика XP — это мое детище, и поэтому она отражает мои собственные страхи. Я боюсь:

- делать бессмысленную работу;
- останавливать проекты из-за того, что я не достиг достаточного технического прогресса;
- делать плохие бизнес-решения;
- иметь дело с плохими техническими решениями, которые сделаны за меня бизнесменами;
- прийти к концу карьеры разработчика программных систем и понять, что было бы лучше, если бы я больше времени проводил с детьми;
- делать работу, которой я не могу гордиться.

XP также отражает вещи, которых я не боюсь:

- кодировать;
- изменять мой взгляд на вещи;
- продолжать работу, ничего не зная о будущем;
- надеяться на других людей;
- изменять анализ и дизайн функционирующей системы;
- писать тесты.

Я должен был научиться не бояться этих вещей. Это не пришло ко мне само собой, особенно если учесть, что так много людей говорили мне о том,

что именно этих вещей и следует бояться и что я должен прилагать все свои усилия для того, чтобы избежать этих вещей.

Ожидание

Один юноша пришел к мастеру фехтования. Когда они сидели на солнышке рядом с хижиной мастера, мастер преподнес молодому человеку свой первый урок: «Вот твой деревянный меч для тренировок. У меня тоже есть деревянный меч, и я могу ткнуть им в тебя в любой момент — при этом ты должен блокировать мой удар». *Тык!*

«Ой!»

«Я же сказал, в любой момент». *Тык!*

«Ой!»

Молодой человек схватил свой деревянный меч и грозно посмотрел на мастера.

«О, я не буду тыкать в тебя сейчас, потому что сейчас ты только этого и ждешь».

В течение следующих нескольких дней ученик постепенно покрывался ссадинами и синяками. Он пытался сосредоточивать свое внимание на всем, что его окружало, но каждый раз, когда его внимание ослабевало, *тык!*

Ученик не мог спокойно есть, он не мог спокойно спать. Он стал параноиком. Он с величайшей осторожностью выглядывал из-за угла и постоянно замирал, пытаясь определить источник малейшего шума. Но каждый раз, когда он в изнеможении опускал глаза или забывал прислушаться, *тпык!*

В скором времени он сел на землю и закричал в отчаянии: «Я больше не могу этого **вынести**! Я никогда не стану фехтовальщиком! Я иду **домой!**» В этот момент, еще не успев понять, что произошло, юноша самопроизвольно выхватил свой меч и молниеносно отразил удар мастера. Тут мастер сказал ему: «Теперь ты готов к обучению».

Мы можем довести себя до безумия благодаря ожиданию. Но подготавливая себя к любому исходу дела, который мы только можем себе представить, мы оставляем себя беззащитными перед неожиданностями, о которых не подумали.

Существует другой путь. Команда может быть отлично подготовлена в любой момент идти в любом из направлений, куда потребуется двигаться бизнесу или системе. Отказавшись от намеренных приготовлений к изменениям, как это ни парадоксально, члены команды становятся полностью готовыми к любым изменениям. Они ничего не ждут. Их ничем невозможно удивить.

Аннотированная библиография

Цель данного раздела — предоставить вам шанс глубже познакомиться с теми аспектами XP, которые заинтересовали вас больше всего[1].

Философия

Sue Bender, *Plain and Simple: A Woman's Journey to Amish,* HarperCollins, 1989; ISBN 0062501860.

Путешествие женщины в Эмиш — больше — это не значит лучше. Меньше — тоже может оказаться не лучше.

Leonard Coren, *Wabi-Sabi: For Artists, Designers, Poets, and Philosophers,* Stone Bridge Press, 1994; ISBN 1880656124.

Ваби-саби: Для художников, дизайнеров, поэтов и философов — XP не нацелена на создание чего-либо, подобного трансцендентальному идеалу в разрабатываемых вами программах. Ваби-саби — это эстетическое торжество всего простого и функционального.

Richard Coyne, *Designing Information Technology in the Postmodern Age: From Method to Metaphor,* MIT Press, 1995; ISBN 02620322887.

Проектирование информационной технологии в эпоху постмодернизма: от метода к метафоре — описывает различия между модернистским и постмодернистским мышлением, тема, которая обыгрывается везде в XP. В этой книге содержится также отличная дискуссия о важности метафор.

Philip B. Crosby, *Quality Is Free: The Art of Making Quality Certain,* Mentor Books, 1992; ISBN 0451625854.

Качество бесплатно: Искусство делать качество несомненным — разбивает модель нулевой суммы для четырех переменных — времени, объема работ, затрат и качества. Вы не можете сделать программу быстрее, снизив ее качество. Напротив, вы можете повысить производительность программы, повысив ее качество.

[1] Многие из упомянутых здесь изданий свободно доступны в Интернете. Я намеренно не даю конкретных ссылок, поскольку месторасположение документов может измениться, но поисковый сервер http://www.google.com выдавал мне ссылку на текст нужной книги, как правило, в числе первых десяти ссылок. — *Примеч. ред.*

George Lakoffand Mark Johnson, *Philosophy in the Flesh: The Embodied Mind and Its Chalange to Western Thought,* Basic Books, 1998; ISBN 0465056733.

Философия во плоти: Воплощенное мышление и его вызов западному образу мысли — хороший анализ метафор и размышлений. В книге также описывается, каким образом метафоры сливаются друг с другом, чтобы сформировать совершенно новые метафоры, как это происходит при разработке программного обеспечения. Старые метафоры, позаимствованные из обычного проектирования, математики и т. п., со временем становятся уникальными метафорами конструирования программных систем.

Bill Mollison, Rena Mia Slay, *Introduction to Permaculture,* Ten Speed Press, 1997; ISBN 0908228082.

Введение в пермакультуру— высокоинтенсивное потребление в западном мире, как правило, ассоциируется с эксплуатацией и истощением ресурсов. Пермакультура — это сельскохозяйственная дисциплина, обеспечивающая постоянное высокоинтенсивное потребление, поддерживаемое синергетическим эффектом нескольких простых методик. Практический интерес представляет идея, что наибольший рост имеет место в местах пересечения различных территорий. Пермакультура максимизирует пересечения и взаимодействия между территориями, смешивая посадки различных пород растений и используя для посадок окрестности озер с сильно изрезанными берегами. XP максимизирует взаимодействие за счет привлечения заказчиков на место разработки и парного программирования.

Отношение

Christopher Alexander, *Notes on the Synthesis of Form,* Harvard University Press, 1970; ISBN 0674627512.

Заметки о синтезе формы — Александр начинает с размышлений о том, что дизайн можно рассматривать как набор решений, направленных на устранение конфликтующих ограничений, затем переходит к рассмотрению других решений, которые направлены на разрешение остающихся ограничений.

Christopher Alexander, *The Timeless Way of Building,* Oxford University Press, 1979; ISBN 0195024028.

Извечный путь строительства — в этой книге описываются взаимоотношения между дизайнерами/строителями и пользователями строений, которые во многом напоминают взаимоотношения между программистами и заказчиками.

Cynthia Heimel, *Sex Tipsfor Girls,* Simon & Schuster, 1983; ISBN 0671477250.

Сексуальные советы для девочек — искренний энтузиазм — это самая основная методика. При наличии искреннего энтузиазма все остальное становится на свои места. Если энтузиазма нет — не стоит и браться.

The Princess Bride, Rob Reiner, director, MGM/UA Studios, 1987.

Принцесса под венец

«Мы никогда не добьемся, чтобы это было живым».

«Вздор. Ты говоришь это потому, что никто никогда не пробовал».

Field Marshal Irwin Rommel, *Attacks: Rommel,* Athena, 1979; ISBN 0960273603.

Наступления: Роммель — любопытные примеры удачных действий в условиях, которые выглядят безнадежными.

Frank Thomas and Ollie Johnston, *Disney Animation: The Illusion of Life,* Hyperion, 1995; ISBN 0786860707.

Анимация Диснея: иллюзия жизни — описание того, как структура команды Диснея эволюционировала в течение нескольких лет в соответствии с изменяющимися бизнесом и технологиями. В книге содержится также множество советов для разработчиков пользовательских интерфейсов, а также несколько действительно крутых картинок.

Внезапные процессы

Christopher Alexander, Sara Ishikawa, Murrey Silverstein, *A Pattern Language,* Oxford University Press, 1977; ISBN 0195019199.

Язык образов — пример системы правил, предназначенных для производства внезапных свойств. Можно спорить о том, являются ли правила удачными или нет, однако сами по себе правила представляют определенный интерес. В книге содержится отличное, но, на мой взгляд, слишком краткое обсуждение дизайна рабочего места.

James Gleick, *Chaos: Making a New Science,* Penguin USA, 1988; ISBN 0140092501.

Хаос: создание новой науки — мягкое введение в теорию хаоса.

Stuart Kauffman, *At Home in the Universe: The Search for Laws of Self-Organization and Complexity,* Oxford University Press, 1996; ISBN 0195111303.

Дома во вселенной: в поиске законов самоорганизации и сложности — несколько более жесткое введение в теорию хаоса.

Roger Lewin, *Complexity: Life at the Edge of Chaos,* Collier Books, 1994; ISBN 0020147953.

Сложность: жизнь на грани хаоса — еще одна теория хаоса.

Margaret Wheatley, *Leadership and the New Science,* Berrett-Koehler Pub, 1994; ISBN 1881052443.

Лидерство и новая наука — отвечает на вопрос: «Что если мы возьмем теорию самоорганизующихся систем в качестве метафоры для менеджмента?»

Системы

Gerald Weinberg, *Quality Software Management: Volume 1, Systems Thinking,* Dorset House, 1991; ISBN 0932633226.

Качественный менеджмент разработки программного обеспечения. Том 1, Размышления о системах — система и нотация для размышлений о системах взаимодействующих действий.

Norbert Weiner, *Cybernetics,* MIT Press, 1961; ISBN 1114239089.

Кибернетика — более глубокое, но несколько более сложное введение в системы.

Warren Witherell and Doug Evrard, *The Athletic Skier,* Johnson Books, 1993; ISBN 1555661173.

Лыжник-спортсмен — система взаимосвязанных правил горнолыжного спуска. Из этой книги я позаимствовал правило 20 на 80.

Люди

Tom DeMarco, Timothy Lister, *Peopleware,* Dorset House, 1999; ISBN 0932633439.

Человеческое программное обеспечение — наряду с книгой *The Psychology of Computer Programming* «Психология компьютерного программирования», эта книга расширяет практический диалог о программах, как продукте человеческой деятельности. Программы в особенности рассматриваются как продукт деятельности групп людей. Из этой книги я позаимствовал принцип «принимаемой ответственности».

Carlo d'Este, *Fatal Decision: Anzio and the Battle for Rome,* Harper-Collins, 1991; ISBN 006092148X.

Судьбоносное решение: Анзио и битва за Рим — что случается, когда эго становится на пути ясного мышления.

Robert Kanigel, *The One Best Way: Frederick Winslow Taylor and the Enigma of Efficiency,* Penguin, 1999; ISBN 0140260803.

Один самый лучший способ: Фредерик Уинслоу Тэйлор и загадка эффективности — биография Тейлора, человека, который разместил свою работу в контексте, который помогал ему увидеть пределы своего мышления.

Gary Klein, *Sources of Power*, MIT Press, 1999; ISBN 0262611465.

Источники силы — простой, читаемый текст о том, как опытные люди в действительности принимают решения в сложных ситуациях.

Thomas Kuhn, *The Structure of Scientific Revolution,* University of Chicago Press, 1996; ISBN 0226458083.

Структура научных революций — для многих XP означает изменение парадигмы. Изменение парадигмы обладает предсказуемыми эффектами. В книге содержится описание некоторых из них.

Scott McCloud, *Understanding Comics,* Harper Perennial, 1994; ISBN 006097625X.

Что такое комиксы — в последних двух главах разговор заходит о том, почему люди пишут комиксы. Прочитав книгу, я задумался о том, почему я пишу программы. В книге содержится также неплохой материал о связи между ремеслом создания комиксов и искусством создания комиксов. Существуют параллели с ремеслом создания программ (тестирование, переработка) и искусством создания программ. В книге вы найдете также неплохие рекомендации для дизайнеров пользовательских интерфейсов, например, сведения о пространстве между элементами дизайна, упаковке информации в небольшие пространства, не создавая при этом беспорядка.

Geoffrey A. Moore, *Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers,* HarperBusiness, 1999; ISBN 0066620023.

Переход через бездну: маркетинг и продажа высокотехнологичных продуктов основным покупателям — изменение парадигмы с точки зрения бизнеса. Различные люди будут готовы принять XP на разных этапах ее эволюционирования. Появление некоторых барьеров легко предсказать. Существуют простые стратегии преодоления этих барьеров.

Frederic Winslow Taylor, *The Principles of Scientific Management,* 2nd ed. Institute of Industrial Engineers, 1998 (1st ed. 1911); ISBN 0898061822.

Принципы научного менеджмента — эта книга стала началом развития «тейлоризма». Специализация и жесткий принцип «разделяй и властвуй» способствуют тому, что большее количество автомобилей производится за меньшую цену. Мой опыт подсказывает мне, что эти принципы не имеют смысла в области разработки программного обеспечения, в них нет никакого смысла как с точки зрения бизнеса, так и с общечеловеческой точки зрения.

Barbara Tuchman, *Practicing History,* Ballantine Books, 1991; ISBN 0345303636.

Практическая история — погруженный в размышления историк думает о том, как он делает историю. Наряду с книгой *Understanding Comics*

«Что такое комиксы» эта книга хорошо показывает, почему вы делаете то, что вы делаете.

Colin M. Turnbull, *The Forest People: Study of the Pygmies of the Congo,* Simon & Schuster, 1961; ISBN 0671640992.

Лесные народы: исследование пигмеев Конго — общество с достаточным количеством ресурсов. Моя мечта — создать подобное ощущение внутри команды.

Colin M. Turnbull, *The Mountain People*, Simon & Schuster, 1972; ISBN 0671640984.

Горные народы — общество с недостаточным количеством ресурсов. Хорошо описывает несколько проектов, в которых я принимал участие. Я ни за что не хочу больше работать в подобной атмосфере.

Mary Walton, W. Edwards Deming, *The Deming Management Method,* Perigee, 1988; ISBN 0399550011.

Метод управления по Демингу — Деминг явно указывает на страх как основной барьер на пути к повышению производительности. Исследование основано на методах статистического контроля качества, однако очень многое зависит от человеческих эмоций и их влияния.

Gerald Weinberg, *Quality Software Management: Volume 4, Congruent Action,* Dorset House, 1994; ISBN 0932633285.

Качественный менеджмент разработки программного обеспечения. Том 4, Конгруэнтное действие — когда вы говорите одно, а делаете другое, происходят плохие вещи. Эта книга о том, как быть последовательным, как распознать непоследовательность в других и что с этим делать.

Gerald Weinberg, *The Philology of Computer Programming,* Dorset House, 1998; ISBN 0932633420.

Психология компьютерного программирования — программы пишутся людьми. Удивительно, не правда ли? Удивительно, что многие этого до сих пор не понимают...

Gerald Weinberg, *The Secrets of Consulting,* Dorset House, 1986; ISBN 0932633013.

Секреты консалтинга — стратегии введения изменений в силу. Книга полезна для инструкторов XP.

Управление проектами

Fred Brooks, *The Mythical Man-Month,* Addison-Wesley, 1995; ISBN 0201835959.

Мифический человеко-месяц — рассказ, который заставит вас задуматься о четырех переменных. В юбилейном издании содержится также лю-

бопытный диалог о знаменитой статье *«No Silver Bullet»* «Не серебряной пулей».

Brad Cox, Andy Novobilski, *Object-Oriented Programming — An Evolutionary Approach*, Addison-Wesley, 1991; ISBN 020158348.

Объектно-ориентированное программирование — эволюционный подход — истоки парадигмы электронного конструирования в области разработки программного обеспечения.

Ward Cunningham, «Episode: A Pattern Language of Competitive Development» in *Pattern Languages of Program Design 2,* John Vlissides ed. Addison-Wesley, 1996; ISBN 0201895277 (http://c2.com/ppr/episodes.html).

«Эпизод: Язык образов (паттернов) при конкурирующей разработке» в книге *«Языки образов (паттернов) в проектировании программ»* — многие идеи XP изначально были отражены в *«Эпизодах»*.

Tom DeMarco, *Controlling Software Projects,* Yourdon Press, 1982; ISBN 0131717111.

Контролирование программных проектов — превосходные примеры создания и использования обратной связи для оценки программных проектов.

Tom Gilb, *Principles of Software Engineering Management,* Addison-Wesley, 1998; ISBN 0201192462.

Принципы менеджмента в области конструирования программного обеспечения — отличный случай эволюционного производства — небольшие версии, постоянная переработка, интенсивный диалог с заказчиком.

Ivar Jacobson, *Object-Oriented Software Engineering: A Case Driven Approach,* Addison-Wesley, 1992; ISBN 0201544350.

Конструирование объектно-ориентированного программного обеспечения: подход, основанный на ситуациях, — отсюда я позаимствовал идею развития разработки на основе историй (use cases — случаи использования).

Ivar Jacobson, Grandy Booch, James Rumbaugh, *The Unified Software Development Process,* Addison Wesley Longman, 1999; ISBN 0201571692.

Унифицированный процесс разработки программного обеспечения — с философской точки зрения я во многом согласен с этой книгой — короткие итерации, концентрация на метафоре, использование историй для управления разработкой.

Philip Metzger, *Managing a Programming Project,* Prentice-Hall, 1973; ISBN 0135507561.

Управление программным проектом — наиболее раннее издание, посвященное менеджменту в области разработки программного обеспечения, которое я смог обнаружить. В книге встречаются блестящие мысли,

однако общая перспектива — тейлоризм. Из 200 страниц книги только два параграфа посвящены обслуживанию и поддержке разрабатываемой системы — это прямая противоположность XP.

Jennifer Stapleton, *DSDM Dynamic Systems Development Method: The Method in Practice,* Addison-Wesley, 1997; ISBN 0201178893.

Динамический метод разработки систем (DSDM). Применение метода на практике — DSDM — это одна из перспектив обеспечения контроля за ускоренной разработкой прикладных программ (Rapid Application Development, RAD), не теряя при этом преимуществ этой технологии.

Hirotaka Takeuchi, Ikujiro Nonaka, «The new product development game» *Harward Business Review* [1986], 86116:137-146.

Игра в разработку нового продукта — ориентированный на достижение консенсуса подход к эволюционной разработке продукта. В статье содержатся интересные идеи относительно масштабирования XP для большего количества программистов.

Jane Wood, Denise Silver, *Joint Application Development, 2* ed, John Wiley and Sons, 1995; ISBN 0471042994.

Совместная разработка приложений (JAD) — последователи JAD и инструкторы XP разделяют одну и ту же систему ценностей — содействовать, не вмешиваясь, предоставлять право действовать тем людям, кто лучше всего знает, как принимать решения. JAD фокусируется на создании документа, содержащего требования, относительно которых заказчики и разработчики соглашаются, что их можно и нужно реализовать.

Программирование

Kent Beck, *Smalltalk Best Practice Patterns,* Prentice-Hall, 1996; ISBN 013476904X.

Лучшие практические образцы программирования на Smalltalk — рекламировать эту книгу не позволяет мне моя скромность.

Kent Beck, Erich Gamma, «Test Infected: Programmers Love Writing Tests» *Java Report,* July 1998, volume 3, number 7, pp. 37-50.

Инфицированные тестами. Программисты любят писать тесты — рассматривается разработка автоматических тестов с использованием JUnit — Java-версии xUnit — системы тестирования кода.

Jon Bentley, *Writing Efficient Programs,* Prentice-Hall, 1982; ISBN 013970251-2.

Разработка эффективных программ — лекарство от болезни «это не будет работать достаточно быстро».

Edward Dijkstra, *A Discipline of Programming*, Prentice-Hall, 1976; ISBN 013215871X.

Дисциплина программирования — программирование как математика. Я был вдохновлен идеей поиска красоты через программирование.

Martin Fowler, *Analisys Patterns*, Addison Wesley Longman, 1996; ISBN 0201895420.

Паттерны анализа — каталог идиом для формирования решений при анализе. Материал усваивается сложнее, чем паттерны проектирования (Design Patterns), однако данная книга во многих отношениях более глубокая, так как паттерны анализа связаны с тем, что происходит на стороне бизнеса.

Martin Fowler, ed. *Refadoring: Impoving the Design of Existing Code*, Addison Wesley Longman, 1999; ISBN 0201485672.

Переработка: улучшение дизайна существующего кода — справочник по переработке кода. Его стоит приобрести, изучить и использовать.

Erich Gamma, Richard Helms, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, 1995; ISBN 0201633612.

Паттерны проектирования: приемы объектно-ориентированного программирования — каталог универсальных идиом проектирования для формирования решений в ходе дизайна системы.

Donald E. Knuth, *Literate Programming,* Stanford University, 1992; ISBN 0937073814.

Грамотное программирование — ориентированный на коммуникацию метод программирования. Поддержка программ, разработанных в соответствии с этим методом, — это настоящее мучение, так как при этом нарушается принцип путешествовать налегке. Все же каждому программисту время от времени приходится писать такую программу.

Steve McConnell, *Code Complete: Practical Handbook of Software Construction,* Microsoft Press, 1993; ISBN 1556154844.

Завершение кода: пособие по практическому конструированию программного обеспечения — исследование о том, какой объем внимания и заботы вы можете с пользой вложить в кодирование.

Bertrand Meyer, *Object-Oriented Software Construction,* Prentice-Hall, 1997; ISBN 0136291554.

Конструирование объектно-ориентированного программного обеспечения — дизайн по контракту — это альтернатива или расширение методики тестирования модулей.

Другое

Barry Boehm, *Software Engineering Economics*, Prentice-Hall, 1981; ISBN 0138221227.

Экономика конструирования программного обеспечения — стандартный справочник о том, сколько стоит программа и почему.

Larry Gonick, Mark Wheelis, *The Cartoon Guide to Genetics*, HarperPerennial Library, 1991; ISBN 0062730991.

Руководство по генетике в картинках — превосходная демонстрация возможностей изобразительного искусства, как среды коммуникации.

John Hull, *Options, Futures, and Other Derivatives*, Prentice-Hall, 1997; ISBN 0132643677.

Опционы, фьючерсы и другие производные — стандартный справочник по ценообразованию в области опционов.

Edward Tufte, *The Visual Display of Quantitative Information,* Graphics Press, 1992; ISBN 096139210X.

Визуальное отображение количественной информации — набор методик передачи численной информации при помощи картинок. Неплохой материал для того, чтобы понять, как лучше всего представить графики метрик. Помимо прочих достоинств книга превосходно издана.

Словарь терминов

Там, где это возможно, XP использует общеупотребительные, общепринятые и широко распространенные термины. Если некоторые используемые в рамках XP концепции в значительной степени отличаются от концепций в других областях знаний, отличие подчеркивается за счет использования нового термина. Далее перечисляются наиболее важные термины из лексикона XP.

Автоматизированный тест (automated test) — тестовый случай, который работает без какого-либо вмешательства со стороны человека. Тест проверяет способность системы выполнять вычисления и получать ожидаемые значения.

Версия (release) — набор историй, которые вместе обладают смыслом с точки зрения бизнеса.

График выполнения работ (commitment schedule) — дата выпуска очередной версии продукта. В ходе каждой итерации график выполнения работ пересматривается. Модификация графика выполняется при помощи переоценки и регенерации.

Заказчик (customer) — роль в команде XP. Заказчик выбирает, какие истории должны быть реализованы в системе, какие из историй необходимо реализовать в первую очередь, а какие могут быть отложены. Заказчик также определяет тесты, благодаря которым можно убедиться в том, что истории корректно выполняются.

Игра в планирование (Planning Game) — процесс планирования в рамках XP. Бизнес должен указать, что должна делать система. Разработчики указывают, сколько стоит каждая возможность и какой бюджет доступен в день/неделю/месяц.

Идеальное время программирования (Ideal programming time) — измерение времени работы во время формирования предварительной оценки. Вы задаете себе вопрос: «Сколько может мне потребоваться времени для того, чтобы сделать это, если меня не будут отвлекать, если не будет никаких неприятностей, форс-мажорных обстоятельств, катастроф и аварий?»

Инженерная задача (engineering task) — некоторая вещь, о которой программист знает, что ее должна делать система. Для реализации задачи необходимо отводить от одного до трех идеальных дней программирования. Большинство задач можно сформулировать напрямую на основании историй.

Инструктор (coach) — роль в команде XP. Инструктор наблюдает за процессом как за единым целым и обращает внимание команды на надвигающиеся проблемы или на возможности улучшить процесс.

Исследование (exploration) — фаза разработки, на которой заказчик сообщает о том, что в общих чертах должна делать система.

История (story) — некоторая вещь, которую по желанию заказчика должна делать система. Работу над реализацией истории можно оценить в период от одной до пяти идеальных недель программирования. Истории должны быть подвержены тестированию.

Итерация (iteration) — период длительностью от одной до четырех недель. В начале этого периода заказчик выбирает истории, которые необходимо реализовать в течение итерации. В конце итерации заказчик может запустить свои функциональные тесты для того, чтобы убедиться, что итерация выполнена успешно.

Менеджер (manager) — роль в команде XP. Менеджер занимается распределением ресурсов.

Партнер (partner) — человек, являющийся вашим напарником при программировании в паре.

Переоценка (reestimation) — ход во время игры в планирование, когда команда заново оценивает все оставшиеся истории, еще не реализованные в рамках текущей версии.

Переработка (refactoring) — внесение в систему изменений таким образом, что при этом поведение системы не меняется, однако улучшаются некоторые ее нефункциональные качества — простота, гибкость, понятность, производительность.

План итерации (iteration plan) — набор историй и набор задач. Программисты выбирают задачи и оценивают их.

Программирование парами (pair programming) — техника программирования предусматривает, что два человека в одно и то же время занимаются программированием одной задачи за одним компьютером, используя одну клавиатуру, одну мышь и один монитор. В XP состав пар меняется, как правило, два раза в день.

Программист (programmer) — роль в команде XP. Программист анализирует, проектирует, тестирует, программирует и интегрирует.

Производство (production) — фаза разработки, на которой заказчик реально использует систему для зарабатывания денег.

Ревизор (tracker) — роль в команде XP. Ревизор измеряет текущее состояние процесса в численном выражении.

Регенерация (recovery) — ход во время игры в планирование, когда заказчик сокращает объем работ над текущей версией продукта, желая сохранить дату выпуска этой версии в случае, если либо предваритель-

ные оценки оказываются заниженными, либо скорость работы команды по тем или иным причинам снижается.

Системная метафора (system metaphor) — история, описывающая логику работы системы, которую могут рассказать любые участники проекта — заказчики, программисты и менеджеры.

Скорость команды (team speed) — количество идеальных недель программирования, которое может быть обеспечено командой за заданный промежуток календарного времени.

Тест модуля (unit test) — тест, написанный с точки зрения программиста.

Тестовый случай (test case) — набор воздействий на систему и соответствующих реакций системы. Воздействия на систему выполняются автоматически, так же автоматически выполняется проверка корректности реакций. Каждый тест должен оставлять систему в том состоянии, в котором она находилась до тестирования, благодаря этому тесты могут выполняться независимо друг от друга.

Фактор (коэффициент) нагрузки (load factor) — измеренное отношение между реальным календарным временем и идеальным временем программирования. Как правило, находится в пределах от 2 до 4.

Функциональный тест (functional test) — тест, написанный с точки зрения заказчика.

Энтропия (entropy) — тенденция системы к накоплению ошибок с течением времени и к существенному росту стоимости внесения в нее изменений.