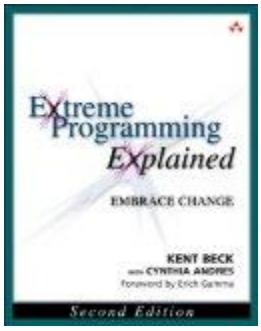
"Экстремальное программирование - 2.0", Микеле Марчези

«В реальном мире даже программисты могут оказаться нормальными людьми. Экстремальное программирование – это возможность проверить себя, быть собой, возможность понять, что все это время проблема была вовсе не в вас, а в том, что вы выбрали себе неправильное окружение».

Кент Бек («Экстремальное программирование», второе издание)



Первое издание книги Кента Бека «Extreme Programming Explained – Embrace Change» (в русском переводе «Экстремальное программирование») увидело свет в октябре 1999 года. Экстремальное программирование (XP) можно любить, можно ненавидеть, а вот отмахнуться как от чего-то несущественного уже нельзя . Эта книга, продававшаяся огромными тиражами и переведенная на десятки языков, открыла дорогу всему многообразию гибких методологий. Гибкие методологии можно применять полностью или частично, не применять вовсе, но ни один серьезный специалист в области программирования не может оставить их без внимания. Теперь каждое руководство по программированию обязательно включает в себя главу о гибких методологиях и экстремальном программировании.

Пять лет спустя Кент Бек опять привлек к себе внимание публикацией второй версии этой книги – на этот раз в соавторстве с Цинтией Андрес (Cynthia Andres). Будьте уверены, речь идет не о простом переиздании с небольшими исправлениями. Во втором издании авторы критически обозревают все, что произошло в этой области за последние пять лет, и практически полностью переосмысливают экстремальное программирование (правда, сохраняя исходные принципы методологии, или по меньшей мере, желание им следовать).

В первом издании книги экстремальное программирование насчитывало четыре основных ценности, пятнадцать базовых принципов и двенадцать практик. Более того, Кент Бек четко и недвусмысленно писал, что экстремальным программированием может считаться только полное и безоговорочное следование всем этим знаменитым двенадцати практикам. Во втором издании никаких двенадцати практик нет уже и в помине! У новой версии экстремального программирования есть пять основных ценностей, четырнадцать принципов, тринадцать главных практик и одиннадцать дополнительных. При этом новые 24 практики лишь отчасти напоминают исходные двенадцать. Две из них, *метафора* и *стандарты написания кода*, и вовсе исчезли.

В этом обзоре я постараюсь кратко описать новые концепции экстремального программирования, а вам, уважаемые читатели, оставлю удовольствие читать оригинал – книгу Кента Бека и Цинтии Андрес «Экстремальное программирование: что делать, когда всё постоянно изменяется», изданную в издательстве Addison Wesley в 2005 году.

Основные ценности

Новая версия экстремального программирования базируется уже на пяти ценностях, которые, как считает Бек, являются главными слагательными успеха любого проекта по разработке программного обеспечения. Основные ценности – это не только руководство по разработке ПО, это еще и источник вдохновения всей методологии. Четыре ценности вы

знаете по первой книге. Теперь к ним добавляется пятая – *уважение*. Итак, вот основные ценности экстремального программирования во втором издании:

Общение: бОльшая часть проблем и ошибок всегда связана с недостатком общения. Следовательно, необходимо максимально увеличить общение между членами команды программистов, а также между программистами и заказчиками. Самым эффективным является прямое общение между людьми. Нельзя также забывать и о другом виде общения – между артефактами и людьми, которые их читают. Все артефакты должны нести адекватную, неустаревшую информацию. Кроме того, их должно быть удобно читать.

Простота: самая рациональная их всех ценностей XP. Заключается она в следующем: «Останавливайся на самом простом решении, которое позволяет выполнить задачу». Однако простое решение (именно простое, а не примитивное) как раз труднее всего найти. Чем проще решение, тем больше за ним стоит опыта, идей и тяжелого труда. Такие решения требуют общения, для них требуется гораздо меньше кода, они улучшают качество программного приложения. Основное требование звучит как «не стоит заранее планировать повторное использование одного и того же решения; чем проще система, тем легче добавлять в нее функциональность по мере необходимости».

Обратная связь: у вас всегда должна быть возможность определить, насколько система, которую вы пишите, далека от необходимого набора функциональности. Лучшие инструменты обратной связи – это непосредственное общение с заказчиком и набор автоматизированных тестов, который растет вместе с системой. С одной стороны, обратная связь является важной составляющей общения: чтобы узнать мнение заказчика, вам надо с ним общаться. С другой, полученные таким образом сведения становятся темой для последующего общения. Обратная связь помогает найти простое решение. Зачастую простые решения достигаются методом проб и ошибок. И опять-таки, чем проще система, тем легче поддерживать обратную связь.

Смелость, кураж: все существующие методологии и процессы разработки предназначены для того, чтобы обуздать и уменьшить наш страх. Чем больше страха мы испытываем перед каким-нибудь проектом, тем серьезнее и «тяжелее» должна быть методология. Общение, простота и обратная связь дают нам возможность смело приступать даже к серьезному рефакторингу системы или к большим изменениям в требованиях. Смелость и кураж сами по себе довольно опасны, но в совокупности с остальными ценностями - это мощнейший инструмент для осуществления больших изменений в системе.

Уважение: все четыре предыщущие ценности подразумевают *уважение* членов команды друг к другу. Если программисты не уважают друг друга и свою работу, то ни одна методология им не поможет. Вы должны проявлять уважение к коллегам по работе, их труду, к вашей компании, а также к тем, в чью жизнь войдет написанное вами приложение. Как видите, в основных ценностях экстремального программирования нет никаких советов относительно того, как руководить проектом или как писать программный код. Это описывается в практиках XP, но прежде, чем перейти к практикам, мы должны остановиться на *принципах*.

Принципы экстремального программирования

В новой версии XP принципы являются как-бы промежуточным звеном между весьма синтетичными и абстрактными ценностями и практиками, в которых даются конкретные указания по разработке ПО. Теперь в XP различают следующие четырнадцать принципов:

Человечность: программное обеспечение создается людьми, поэтому человеческий фактор остается основным ключом к разработке качественного программного продукта. Экстремальное программирование ставит своей целью выгоду как отдельных людей, так и целых организаций, чтобы польза от использования методологии ощущалась обеими

сторонами. Конечно же, здесь нужен баланс: если мы переоценим нужды коллектива разработчиков, это может привести к снижению продуктивности труда, люди не будут работать подобающим образом, что приведет компанию к убыткам. А убытки, в свою очередь, могут привести к закрытию проекта или даже всей фирмы, что больно ударит по людям, которые в ней работали. Если же вы переоцените нужды организации, люди начнут перерабатывать, между ними участятся конфликты. Это тоже приведет компанию к потерям. В новой книге Кент приводит следующий список потребностей команды:

- Базовая безопасность – необходимость справляться с работой;
- Достижения – чувство собственной полезности на работе;
- Принадлежность – способность причислять себя к группе;
- Рост – возможность расширять и углублять свои навыки, видеть новые перспективы;
- Близость – способность понимать других и быть понятым.

Economics: программное обеспечение, которое вы производите, должно обладать некой ценностью (business value). В XP есть два ключевых экономических момента: ценность программного продукта на текущий момент и ценность дополнительных возможностей (value of options). Согласно первой, доллар, заработанный сегодня, стоит больше, чем доллар заработанный завтра, поэтому чем быстрее мы выпускаем программный продукт и начинаем зарабатывать деньги, тем больше прибыль. Это напрямую связано с ценой дополнительных возможностей программы. Так, если вы можете отложить архитектурные изменения до того момента, когда их необходимость станет совершенно очевидной, то сбережете своей компании большие деньги. Практики XP приветствуют инкрементальный дизайн, внимание к тому, что приносит выгоду клиенту, и отношение к разработке, которое можно было бы охарактеризовать словами «плати за то, чем пользуешься». Все это значительно упрощает процесс принятия решений.

Взаимная выгода: всякая деятельность должна приносить выгоду задействованным людям и организациям. Это, пожалуй, сам важный из принципов XP, хотя его очень сложно придерживаться. Из любой проблемы легко найти выход, при котором пострадает одна из взаимодействующих сторон. И нередко нам очень хочется идти именно таким путем. Однако подобные решения всегда ухудшают положение, так как они портят отношения и разрушают рабочую среду. Чтобы не увеличивать количество проблем, вам понадобятся навыки, которые обеспечат выгодное сотрудничество и вам, и вашим клиентам, причем и сейчас, и в будущем.

Сходство: в природе часто встречаются фрактальные структуры, очень похожие между собой, но существующие на разных уровнях. Точно такие же принципы можно применить и к разработке программного обеспечения: мы можем использовать одни и те же идеи для решения проблем, возникающих в разных контекстах. Например, одна из основных составляющих методологии XP состоит в том, чтобы сначала написать тесты, которые заведомо не будут проходить, а уже потом – написать код, после которого тесты пройдут успешно. То же самое можно «масштабировать» на разные уровни временной шкалы: в течение целого квартала вы составляете перечень задач, которые должно решать приложение, а потом короткие «рассказы», которые описывают их более подробно. В начале недели вы выбираете те «рассказы», где описаны задачи, над которыми вы будете работать в течение этой недели, а уже потом пишете тесты приемки (acceptance tests) и, в конце концов, приступаете к программному коду, благодаря которому эти тесты будут работать. Если сузить временной промежуток до нескольких часов – вы сначала пишетеunit тесты, а затем код, который обеспечивает их выполнение.

Все лучше и лучше: постоянное развитие и движение вперед – ключ к пониманию XP. Конечно, совершенство недостижимо, однако надо постоянно к нему стремиться. Каждый день надо стараться работать как можно лучше и придумывать новые способы сделать работу еще более эффективной. Таким образом, с каждой итерацией продукт становится

все лучше и лучше – как с точки зрения качества кода, так и с точки зрения функциональных возможностей. Это обеспечивается за счет отзывов заказчика, автоматических тестов, и конечно, самой команды разработчиков.

Разнообразие: если все члены команды похожи друг на друга, работа в ней может оказаться очень комфортной, но малоэффективной. Лучше, когда в команде есть представители из разных областей знаний, разные характеры. Это позволяет лучше находить и решать проблемы. Конечно, в такой команде могут возникать конфликты, которые придется как-то решать. Однако если вы способны разрешать конфликтные ситуации и определять лучшее из альтернативных мнений, отдавайте предпочтение «неоднородным» командам. Они умеют находить неожиданные решения в сложных ситуациях, что в нашей области очень важно.

Обдумывание: эффективно работающие программисты не просто пишут код. Они спрашивают себя, *как* они работают и *почему* они делают данную систему именно так, а не иначе. Людям нужно четко видеть причины, стоящие за успехом (или провалом) их продукта. Не надо прятать ошибки. Куда полезнее открыто признать их и попытаться вынести из них полезный урок на будущее. Во время ежеквартальных и еженедельных итераций отведите некоторое время на обсуждение того, как движется проект, и какие улучшения можно было бы внести в процесс разработки. Но не надо слишком уж заострять на этом внимание. В нашей индустрии есть много примеров того, как программисты настолько переключались на совершенствование процесса работы, что на саму работу у них не оставалось времени! Сначала идет работа – потом обдумывание – потом снова работа.

Течение (flow): в данном случае, это означает постоянную размеренную разработку качественного программного обеспечения, включающую в себя все соответствующие виды деятельности. В методологии XP принято считать, что все виды деятельности по разработке ПО должны протекать непрерывно, подобно потоку. Этим она отличается от других методологий, где разработка распадается на последовательность определенных фаз, последней из которых является выпуск программного продукта. Только благодаря непрерывному течению разработки программисты могут рано узнать мнение заказчиков о системе, получить уверенность, что работа ведется в должном направлении, а кроме того, избежать трудностей последней интеграции, этого вечного «трам-тарарама» перед выпуском продукта.

Новые возможности: в проблемах надо видеть прежде всего возможность что-то улучшить. Проблемы неизбежны, но чтобы достичь совершенства мало просто «решить проблему». Надо превратить проблему в возможность узнать что-то новое, найти лучшее решение. Может быть, у вас не получается строить долгосрочные планы? Тогда попробуйте составить планировать ежеквартально, и каждые три месяца корректируйте то, что напланировали. В вашей команде есть программист, который делает слишком много ошибок? Посадите его программировать в паре с гуру! Практики методологии XP доказали свою эффективность именно тем, что решали старые проблемы, существовавшие, пожалуй, со времени появления самой отрасли производства ПО.

Избыточность: действительно сложные или критичные для проекта проблемы надо решать несколькими способами одновременно. В этом случае, если одно решение окажется несостоятельным, другие могут помочь предотвратить катастрофу. В таких случаях в итоге избыточность с лихвой окупается. Проблемы, связанные с функциональностью программного обеспечения, нужно искать и решать различными способами: парным программированием, автоматизированными тестами, работой в общем помещении, активным вовлечением заказчика, и т.д. Разумеется, в этом есть некоторая избыточность – многие недостатки будут обнаруживаться по нескольку раз. Однако качество продукта –

самая главная цель – все окупит. Впрочем, если с помощью какой-то практики удается обнаружить только уже известные дефекты, ее надо отменять за ненадобностью.

Неудачи: если что-то не получается, не бойтесь неудач. Не знаете, как лучше написать часть новой функциональности? Попробуйте сделать это тремя-четырьмя разными способами. Даже если все окажутся неудачными, вы многому научитесь. Полезны ли неудачи? Да, если мы выносим из них ценные уроки. Поэтому не бойтесь неудач. Куда хуже откладывать что-то до последнего момента, пытаясь найти единственно верное решение.

Качество: качество всегда должно быть на высоте. Выпускать продукт низкого качества, чтобы сэкономить средства или время, - заведомо неправильное решение. Как раз напротив, работа над качеством системы способствует улучшению других его свойств, например, производительности и эффективности. Не стоит, однако, путать качество и перфекционизм. Если вы откладываете активную фазу разработки, в надежде найти идеальное решение, вы не будете продвигаться вперед. Гораздо эффективнее попробовать какой-то вариант, выяснить, в чем его недостатки, и быстро их исправить.

Маленькими шажками: если долго готовить серьезные большие изменения, а потом внести их в систему «одним махом», весь проект может оказаться под угрозой срыва. Гораздо надежнее продвигаться вперед маленькими шажками – пусть шаги эти невелики, зато вы можете быть уверены, что проект движется в нужном направлении. Кроме того, маленькими шажками можно двигаться довольно быстро: за короткое время команда программистов может сделать очень много таких «шажков», при этом постоянно получать отзывы и корректировать продвижение работ. Еще один плюс небольших изменений состоит в том, что небольшое изменение, как правило, может привести лишь к небольшим проблемам. А вот чем больше шаг и чем серьезнее изменения, тем больший потенциальный вред может он принести всему проекту.

Ответственность: ответственность можно взять на себя только в добровольном порядке. Конечно, любой начальник может приказать программисту: «Делай то, делай это», но такой подход не работает. Вы неизбежно будете требовать или гораздо меньше того, что нужно, или – что случается чаще – гораздо больше того, что может данный программист. Поэтому получая указания, человек сам должен принять решение: берет ли он на себя ответственность, или же пусть этой проблемой занимается кто-то другой.

Практики XP

Новая версия экстремального программирования насчитывает тринадцать основных практик и одиннадцать дополнительных. Вначале нужно применить основные практики, причем каждая из них может соответствующим образом улучшить процесс разработки. Только после этого можно приступать к дополнительным практикам, которые требуют опыта работы с основными практиками, и практически неприменимы без них.

В целом же можно сказать, что все 24 практики очень важны для процесса разработки, и должны быть применены полностью. Только так проект получит выгоду от экстремального программирования.

Сам Кент Бек никак не классифицирует практики, однако мне кажется уместным разделить их на четыре категории:

- Анализ требований и планирование;
- Команда и человеческий фактор;
- Проектирование;
- Программирование и выпуск продукта.

Обращаю ваше внимание на то, что многие практики можно было бы отнести сразу к нескольким категориям. Так, «парное программирование» значится у меня в группе «Команда и человеческий фактор», однако с тем же успехом ее можно было бы поместить в категорию «Программирование и выпуск продукта». Далее я опишу каждую практику лишь один раз – в той категории, которая показалась мне наиболее уместной.

Основные практики

Анализ требований и планирование

- **«Рассказы»**: функциональность приложения описывается короткими «*рассказами*», в которых работа системы изложена с точки зрения заказчика. Эти «рассказы» также являются основной движущей силой разработки приложения.
- **Еженедельный цикл**: вся разработка проекта происходит в виде череды еженедельных циклов. В начале недели происходит собрание, на котором заказчик выбирает, какие «рассказы» надо сделать за эту неделю.
- **Ежеквартальный цикл**: планирование в большем масштабе происходит каждый квартал. Оно состоит из обсуждений работы команды и темпов разработки.
- **Слабина**: избегайте обещаний, которые не сможете выполнить. В любой план включайте задачи, которые вы сможете выкинуть, если не будете укладываться в срок. В этом случае у вас будет выход даже в случае непредвиденных проблем.

Команда и человеческий фактор

- **Работа в одном помещении**: команда разработчиков должна сидеть в одном большом помещении – это облегчает общение.
- **Команда как одно целое**: команда должна состоять из людей, обладающих всеми необходимыми для проекта навыками и знаниями. Всех их должно объединять чувство принадлежности общему делу, они должны всячески поддерживать друг друга.
- **Информативность окружения**: в рабочем помещеним должны быть информативные постеры и прочие наглядные пособия, которые показывали бы статус проекта и другую информацию о проделанной работе.
- **Энергичная работа**: люди не должны быть истощены работой, им надо сохранять свежесть и энергичность, чтобы фокусироваться на задачах и уметь эффективно их решать. Следовательно, надо жестко ограничить сверхурочную работу, так чтобы у каждого оставалось время и на личную жизнь. В прежней версии методологии это называлось «приемлимый темп разработки».
- **Парное программирование**: код всегда пишут два программиста, сидящих за одним компьютером.

Проектирование

- **Инкрементное проектирование**: согласно XP, не следует заниматься подробным проектированием системы в самом начале работ. Вместо этого команда разработчиков старается как можно скорее начать писать программный код, чтобы получить отзывы пользователей о системе, и улучшать ее по ходу дела. Конечно, чтобы написать хороший код, система должна быть должным образом спроектирована. Этого XP не отрицает.Вопрос лишь – *когда* заниматься проектированием. Согласно экстремальному программированию, проектированием должно происходить инкрементально во время написания программного кода. Особенно полезно делать это, чтобы убирать ненужное дублирование.
- **Сначала тесты**: перед тем, как редактировать старый код или писать новый, нужно написать тесты, которые будут его проверять. Это поможет решить четыре проблемы:

- «Программирование по-ковбойски»: во время написания кода так легко увлечься и начать писать код для всех задач подряд, которые приходят на ум. Если же сначала написать тесты, которые затем будут проверять код, нам волей-неволей придется сосредоточиться на задаче, которую мы пытаемся решить, а также проверить, насколько правильно мы спроектировали данную часть системы.
- Слаженность и единство: если написать тест трудно, значит, у вас проблемы с дизайном системы, а не с тестированием или программированием. Когда программный код разбит на функционально связанные модули с минимальным количеством двусторонних зависимостей между ними, тестировать его не составит большого труда.
- Доверие: если вы пишете код, который работает, и документируете его с помощью автоматизированных тестов, ваши коллеги будут доверять вам.
- Ритм: во время программирования можно легко увлечься и блуждать в коде в течение нескольких часов. Если вы приучите себя к ритму «тест, код, рефакторинг, тест, код, рефакторинг», этого никогда не случится.

Программирование и выпуск продукта

- **Десятиминутная сборка**: систему должно быть можно собрать (с учетом прогона всех тестов) за десять минут. Это позволит часто повторять операцию и получать отзывы о разрабатываемом продукте.
- **Постоянная интеграция**: разработчики должны выкладывать в репозиторий результаты своей работы каждые два часа, чтобы избежать проблем с интеграцией нового кода.

Дополнительные практики

Анализ требований и планирование

- **Непосредственное вовлечение заказчика**: люди, для которых вы пишете программный продукт, должны стать частью команды и вносить свою лепту в ежеквартальное и еженедельное планирование.
- **Инкрементная поставка продукта**: когда вам предстоит целиком сменить существующую систему, начинайте процесс с изменения нескольких функций, и так постепенно замените всю систему. Избегайте подхода, который можно выразить словами «Все или ничего!»
- **Контракт с оговоренным объемом работ**: контракт на разработку программного обеспечения включает в себя время, затраты и качество системы, однако точные объемы этой системы надо оговаривать в процессе работы. Заключив с заказчиком серию небольших контрактов, можно значительно снизить риски.
- **Плата за использование**: обычно заказчик платит за каждый выпуск программного продукта. Это нередко дает повод для конфликтов между разработчиками и заказчиком, который хочет внести как можно больше новой функциональности в наименьшее количество выпусков продукта. Если исчислять деньгами непосредственную работу над функциональностью, заказчик будет получать более точную и своевременную информацию, и сможет точнее направлять разработку продукта.

Команда и человеческий фактор

- **Постоянство**: команда разработчиков должна работать в одном и том же составе на протяжении нескольких проектов. Те связи, которые возникают между людьми, воистину бесценны, поэтому старайтесь перераспределять людей как можно реже.
- **«Усушка и утряска»**: если команда становится все более продуктивной, не увеличивайте ее нагрузку. Оставьте объем работ прежним, но выделите свободных членов этой команды с тем, чтобы они создавали свои собственные новые команды.

Проектирование

- **Анализ причин и следствий**: каждый раз, когда вы находите ошибку в системе, исправляйте не только ее, но и ее причину. В противном случае эта ошибка может повториться в будущем.

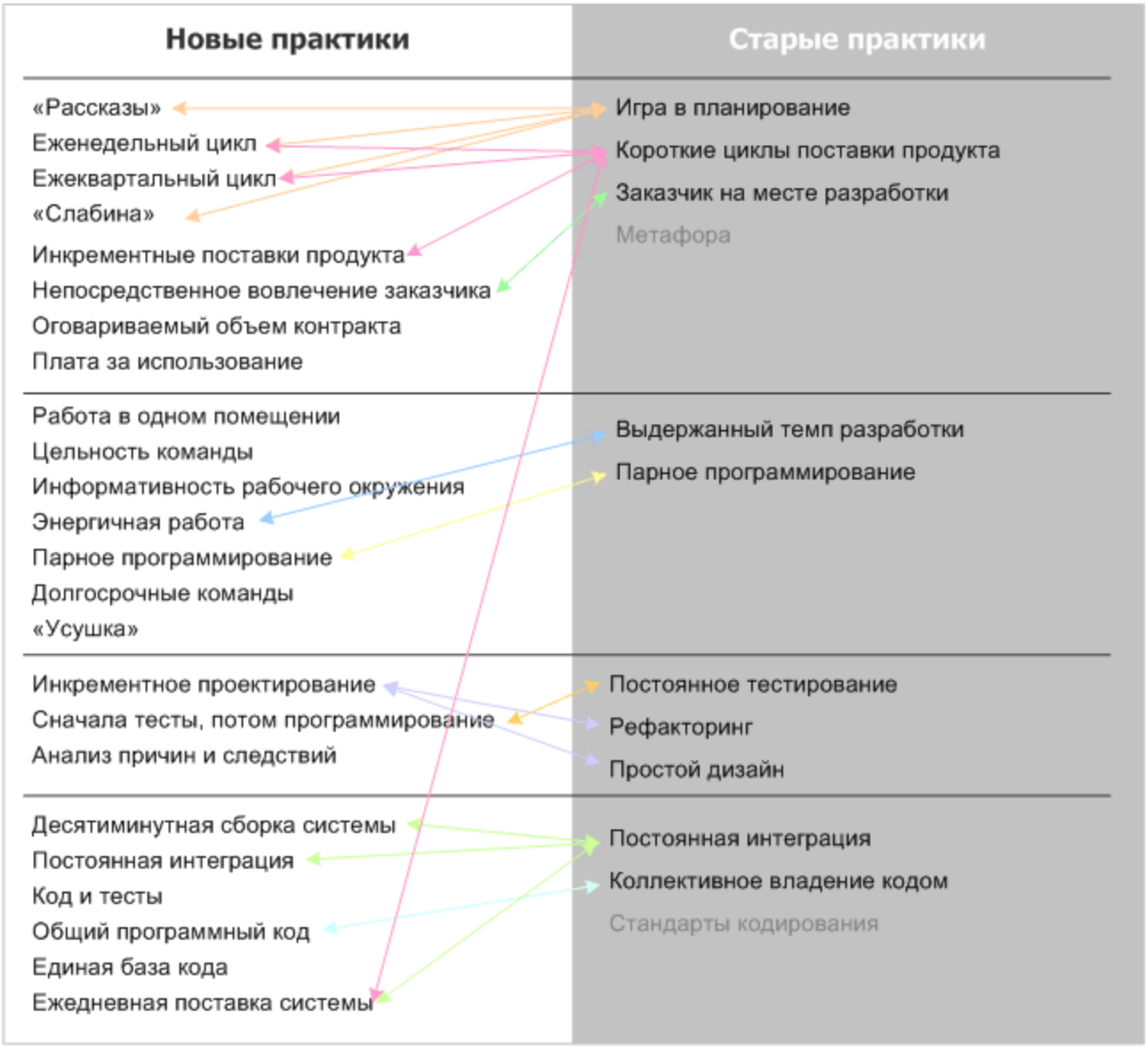
Программирование и выпуск продукта

- **Код и тесты**: только программный код и тесты являются постоянными артефактами системы, которые надо сохранять. Все остальное может быть получено из программного кода и тестов.
- **Общий код**: любой член команды может в любой момент изменить любую часть системы. Эта практика называлась в прежней версии XP «*коллективное владение кодом*».
- **Единая база программного кода**: существует только одна *официальная* версия разрабатываемой системы. Если вам понадобилось создать для чего-то ее ветку, оставляйте ее лишь на несколько часов.
- **Ежедневная поставка системы**: каждую ночь надо собирать новую версию системы и вводить ее в действие. Чем больше разрыв между официальной версией системы и той, что находится у вас в компьютере, тем это рискованнее и дороже для проекта.

Как вы уже, наверное, заметили, в новой версии XP не нашлось места для нескольких изначальных практик, например *стандартов написания кода*. Дело в том, что она стала уже настолько общепринятой, что ее не надо упоминать в рамках этой методологии. Исчезла также и *метафора*, пожалуй, самая сложная и неопределенная из всех двенадцати практик первой версии XP, которую весьма непросто было воплотить в жизнь.

Теперь, когда вы получили первое представление о новых двадцати четырех практиках, можно попытаться проанализировать связь между ними и исходными двенадцатью практиками. Некоторые из новых практик восходят к старым и расширяют их, другие – совершенно новые. К примеру, старая практика «*игра в планирование*» исчезла и превратилась в целых четыре – «рассказы», «еженедельный цикл», «ежеквартальный цикл» и «слабина».

Ниже я привожу таблицу, на которой изображены старые и новые практики XP. Все они разнесены по категориям. Так легче показать связь между старыми и новыми практиками, и так проще понять, чем же новое экстремальное программирование отличается от старого.



И в заключение я хотел бы сказать, что постарался максимально ясно – как обещает книга Кента Бека – изложить суть «гибкой» методологии разработки ПО. Новое экстремальное программирование сильно отличается от того, что было описано в первой книге Кента. Каждый принцип XP, изложенный в первой версии, подвергается сомнению и основательно перерабатывается. Это закономерная эволюция методологии, прошедшей пятилетний путь развития с момента выхода первой книги. В этой книге много новых идей, практик и принципов, за которые некоторые последователи, возможно, даже обвинят Бека в ереси. А может быть, и прислушаются к его советам и сначала все попробуют применить его новые идеи в своих проектах. Впрочем, мне кажется, что Бек никогда не стремился превратить свои книги в «задачник с ответами», которым вы должны безукоснительно подчиняться. Нет, он стремится начать дискуссию – открытое обсуждение того, как надо разрабатывать программное обеспечение, предлагает каждой компании разработать свою собственную версию экстремального программирования, наиболее подходящего для этой компании, ее опыта и контекста ее работы.