ЧАСТЬ I

Быстрая разработка ПО

Взаимодействие между людьми носит сложный характер, а различные вызванные им эффекты всегда “прозрачны”, но оно имеет наибольшее значение среди других рабочих аспектов.

Том Де-Марко и Тимоти Листер, Peopleware, с. 5

Принципы, шаблоны и применяемые практики имеют значение, но вся работа выполняется людьми. Согласно Алистеру Кокберну (Alistair Cockburn)[5], "процессы и технологии оказывают второстепенное влияние на результат выполняемого проекта. Первостепенное влияние оказывают люди".

Конечно, мы не можем управлять группами программистов подобно тому, как они управляют сформированными из компонентов системами, управляемыми процессами. Люди не являются "подключаемыми программными модулями"[6]. Если проекты выполняются успешно, следовательно, группы исполнителей являются самоуправляемыми, а между их членами хорошо налажено сотрудничество.

Компании, придерживающиеся подобных принципов при формировании команд программистов, получают *огромное* преимущество в конкурентной борьбе по сравнению с темы фирмами, которые рассматривают программистов как некую "серую массу". На самом деле сплоченная группа программистов — наиболее мощная сила, влияющая на выполнение задач по разработке ПО.

[5] Частные коммуникации.

[6] Термин, введенный Кеном Беком (Kent Beck).

1

Быстрая разработка ПО

> Флюгер на церковном шпиле быстро выйдет из строя под напором штормового ветра, если не обучится благоразумному искусству уклоняться от любого ветра.
>
> Генрих Гейне

Многим из нас приходилось испытывать на себе трудности работы с проектом, когда отсутствуют четкие инструкции по его исполнению. В этом случае возможны непредсказуемые последствия, повторяются ошибки, а также напрасно затрачиваются усилия. В результате заказчики выражают недовольство из-за смещения графиков проекта и ухудшения его качества. Разработчики впадают в уныние, наблюдая обратную зависимость между объемом трудозатрат и качеством разрабатываемого программного обеспечения.

После такого фиаско возникает ощущение возможности его повторения. Именно это обстоятельство стало причиной разработки *процесса*, ограничивающего действия разработчиков и требующего получения определенных артефактов и результатов. Ограничения и результаты такого рода "извлекаются" из предыдущего опыта, "выбираются" положительные моменты, связанные с выполнением предыдущих проектов. Авторы книги выражают надежду, что вы все же снова приступите к работе, преодолев синдром боязни повторных ошибок.

В силу сложности отдельных проектов, несколько простых ограничений и артефактов не смогут гарантировать отсутствие ошибок. Дефекты и ошибки неизбежны, поэтому задача разработчика заключается в том, чтобы выполнить необходимую диагностику, а также определить дополнительные артефакты и ограничения, позволяющие предотвратить появление подобных ошибок в будущем. Итогом проведения подобного анализа в случае большого количества проектов может являться столь громоздкий процесс, который в значительной мере затруднит выполнение любого проекта.

Большой и громоздкий процесс может послужить причиной возникновения проблем, которые он должен устранять. В результате замедляется работа команды разработчиков, что, в свою очередь, приводит к смещению рабочего графика, а также к раздуванию бюджета. При этом способность команды к обратному реагированию может "приближаться к нулю", из-за чего создаваемый программный продукт не будет отвечать поставленным требованиям. К сожалению, это приводит к тому, что у многих команд разработчиков создается впечатление о недостаточной продуманности самого процесса. Мотивируя неудачи громоздкостью и недостаточной продуманностью процесса, тем самым они способствуют "раздуванию" этого процесса.

Громоздкость и непродуманность процесса — это удачно подобранный термин, описывающий положение дел в большинстве компаний, занимающихся разработкой программного обеспечения 2–3 года назад. Хотя в то же самое время многие

разработчики вообще не использовали какой-либо определенный процесс, реализация на практике громадных процессов становится все более распространенной, и особенно это справедливо в случае больших корпораций (см. приложение С.)

Альянс специалистов по быстрой разработке ПО

В начале 2001 года группа экспертов в области разработки ПО обратила внимание на тот факт, что команды разработчиков во многих корпорациях попали в ловушку постоянно разбухающего процесса. Поэтому они организовали встречу, чтобы сформулировать оценку и принципы, позволяющие командам разработчиков оптимизировать свою работу и своевременно реагировать на изменения. Эта группа придумала себе запоминающееся название: *"Альянс специалистов по быстрой разработке ПО"* (*Agile Alliance*)[1]. Работа над выработкой применяемых на практике оценочных принципов продолжалась на протяжении нескольких последующих месяцев. В результате появился *"Манифест альянса специалистов по быстрой разработке ПО"* (*The Manifesto of the Agile Alliance*).

Манифест альянса специалистов по быстрой разработке ПО

Мы находимся в процессе поиска более эффективных методов разработки ПО, а также помогаем делать это другим. В качестве предмета оценочного анализа выступает следующее:

- индивиды и взаимодействия, связанные с процессами и инструментальными средствами разработки;
- рабочий программный продукт и полный комплект документации;
- совместная работа с заказчиком и обсуждение условий контракта;
- реакция на происходящие изменения и соблюдение плана.

При этом отдается должное компонентам, перечисленным в левой части, но мы все же считаем, что правосторонние компоненты имеют большее значение.

Кен Бек (Kent Beck), Майк Бидль (Mike Beedle), Ари Ван-Биннекум (Arie van Bennekum), Элистер Кокберн (Alistair Cockburn), Уорд Каннингем (Ward Cunningham), Мартин Фаулер (Martin Fowler), Джеймс Гриннинг (James Grenning), Джим Хайсмит (Jim Highsmith), Эндрю Хант (Andrew Hunt), Рон Джеффрис (Ron Jeffries), Джон Керн (Jon Kern), Брайан Мэрик (Brian Marick), Роберт К. Мартин (Robert C. Martin), Стив Мэллор (Steve Mellor), Кен Шваубер (Ken Schwaber), Джефф Сатерлэнд (Jeff Sutherland), Дэйв Томас (Dave Thomas)

[1] Web-узел `www.agilealliance.org`.

Индивиды в аспекте взаимодействия с процессами и инструментальными средствами

Успех в любом деле определяется людьми. Хорошо продуманный процесс не спасет проект от провала, если в команде отсутствуют сильные игроки. С другой стороны, плохо организованный процесс может привести к тому, что деятельность группы профессиональных разработчиков завершится крахом в случае, если они не смогут работать вместе.

Сильный игрок вовсе не обязательно является первоклассным программистом. Его успехи в программировании могут быть чрезвычайно скромными, главное — это умение успешно работать с другими членами команды. На самом деле, эффективное взаимодействие с сотрудниками, коммуникации и общение имеют большее значение, чем талант программирования сам по себе. Команда, состоящая из посредственных программистов, между которыми на должном уровне налажено взаимодействие, имеет больше шансов на успех, чем группа высококлассных программистов, которым не удается наладить работу в составе команды.

Правильно подобранные инструментальные средства также могут стать определяющими в достижении успеха. Компиляторы, интегрированные среды разработки (IDE, Integrated Development Environment), системы проверки исходного кода и некоторые другие компоненты играют жизненно важную роль, обеспечивая команде разработчиков надлежащие условия для работы. Однако значение инструментальных средств может быть преувеличено. Избыток больших и громоздких инструментов так же плохо, как и их недостаток.

Воспользуйтесь тактикой “малых шагов”. Не стоит делать вывод о том, что вы переросли возможности какого-либо инструментального средства до тех пор, пока не опробуете его в деле, после чего сделаете вывод относительно его непригодности для дальнейшей работы. Вместо того чтобы приобретать наиболее мощную и дорогую систему проверки исходного кода, следует найти ее бесплатно распространяемую версию и пользоваться ею до тех пор, пока не выявится несоответствие заявленным требованиям. Перед приобретением лицензии на использование лучших CASE-инструментов, используйте “белые доски” и миллиметровую бумагу в целях обоснования своих аргументов. Прежде чем отдать предпочтение сложнейшей системе баз дынных, попробуйте использовать в работе однородные файлы. Не следует думать о том, что более громоздкие и совершенные инструментальные средства оптимизируют вашу работу автоматически. Чаще всего в этом случае проявляются недостатки, а не какие-либо преимущества.

Помните о том, что формирование команды намного важнее, чем создание среды разработки. Многие команды и менеджеры допускают ошибку, формируя изначально среду разработки и ожидая, что сплоченная команда образуется авто-

матически. Сначала следует сформировать команду, а затем позволить участникам команды сконфигурировать среду разработки, исходя из их собственных потребностей.

Рабочий программный продукт и исчерпывающая документация

Поставка программы без сопровождающей документации может привести к негативным последствиям. Программный код сам по себе не является идеальным средством, позволяющим взаимодействовать по законам логически непротиворечивой и структурированной системы. Настоятельно рекомендуется разработать документацию, удобную для восприятия человеком, в которой описывается система, а также приводится логическое обоснование принимаемых в ходе осуществления проекта решений.

Следует учитывать тот момент, что избыток документов далеко не всегда благо. Огромный объем сопровождающей документации потребует очень много времени на разработку и еще больше времени и трудозатрат на последующую поддержку, отображающую изменения в коде. Если документы не синхронизированы с кодом, они превращаются в громоздкие логически противоречивые источники неправдивой информации и вряд ли смогут применяться на практике.

В любом случае команде следует сформировать и поддерживать логически непротиворечивую и структурированную документацию, сопровождающую программный продукт. Этот документ должен быть *кратким* и *четко сформулированным*. Термин “краткий” подразумевает то, что документ должен состоять максимум из двух десятков страниц. Термин “четко сформулированный” означает, что документ включает полное логическое обоснование проекта и структуры высшего уровня, входящей в состав системы.

Если в нашем распоряжении оказывается краткий логически непротиворечивый и структурированный документ, каким же образом можно обучать новых членов команды работе с системой? Для этого следует хорошо поработать с новыми людьми. Новичков потребуется превратить в полноценных членов команды путем непосредственного обучения и взаимодействия.

Документы, которые являются наиболее подходящими в учебных целях, — это программный код и сама команда. Код не может являться источником ложной информации относительно выполняемых им функций. Выборка логического обоснования и цели из кода может быть связано с определенными трудностями, но все же код сам по себе — это единственный однозначный источник информации. Все члены команды хранят в памяти постоянно изменяющуюся карту-схему системы. Наиболее быстрый и эффективный способ передачи сведений о подобной карте другим членам команды заключается в непосредственном общении и взаимодействии.

Многие команды разработчиков идут по ошибочному пути, сосредотачиваясь на разработке подробной документации вместо того, чтобы нацелить все силы

и средства на разработку самого программного продукта. Зачастую эта ошибка является роковой. На сей счет существует простое правило, именуемое “Первый закон Мартина о документации”.

Не создавайте документацию до тех пор, пока потребность в ней не приобретет первостепенную важность.

Совместная работа с заказчиком и обсуждение условий контракта

Процедура заказа программ весьма отличается от действий, предпринимаемых в процессе заказа обычного товара. Недостаточно просто составить описание требуемого программного продукта, а затем попросить кого-либо разработать его в рамках четко определенного графика за фиксированную цену. Во многих случаях попытки трактовать подобным образом программные проекты терпят полный крах. Иногда эти провалы являются весьма болезненными.

Менеджеры компании зачастую ограничиваются изложением команде разработчиков своих собственных требований, а затем ожидают от них поставки готовой системы, соответствующей поставленным требованиям. Именно подобный метод работы может привести к созданию низкокачественных программ, что эквивалентно краху проекта.

Успешные проекты предполагают регулярную и повторяющуюся обратную связь со стороны заказчиков. Вместо того чтобы полагаться на контракт или отчет о проделанной работе, заказчик программного продукта тесно работает с командой разработчиков, обеспечивая тесную обратную связь, которая позволяет согласовывать достигнутые ими результаты.

Контракт, определяющий требования, график и затраты на проект, как правило, включает массу недостатков. В большинстве случаев определяемые контрактом условия становятся бессмысленными прежде, чем проект будет выполнен[2]. Наилучшие контракты лишь обусловливают способ, с помощью которого команда разработчиков и заказчик будут совместно работать над проектом.

В качестве примера можно привести контракт, который был заключен автором в 1994 году. Этот контракт предполагал выполнение большого проекта, рассчитанного на много лет и включающего до полумиллиона строк программного кода. Условиями контракта предусматривалось, что команда разработчиков получала относительно небольшую ежемесячную ставку. Большие денежные суммы (премии) выплачивались лишь после готовности больших программных блоков заданного размера, реализующих определенные функции системы. Эти блоки детально не описывались в контракте. Здесь просто указывалось, что выдача премий за готовый программный блок производилась лишь после того, когда он проходил через сито приемочных испытаний у заказчика. Детали проведения подобных испытаний в контракте не оговаривались.

[2]Иногда это происходит еще до подписания контракта!

В ходе выполнения проекта разработчики постоянно взаимодействовали с заказчиком. Отдельные программные блоки сдавались практически еженедельно (по пятницам). До понедельника или вторника последующей недели у заказчика накапливался для нас перечень изменений, которые было необходимо внести в программный продукт. Эти изменения распределялись в приоритетном порядке, и их внесение в продукт планировалось на последующие недели. Сотрудничество с заказчиком было настолько тесным, что никогда не возникал вопрос о проведении приемочных испытаний. Заказчик четко представлял, в каком случае блок, кодирующий функции системы, соответствовал сформулированным им требованиям, поскольку имел возможность наблюдать за процессом его разработки еженедельно.

Требования, предъявляемые к данному проекту, непрерывного изменялись. Большинство изменений не носили исключительный характер. Некоторые блоки, кодирующие функции системы, удалялись и замещались новыми программными модулями. Но и контракт, и проект не потерпели крах и оказались успешными. Ключ к такому успеху заключался в тесном сотрудничестве с клиентом, а также в контракте, который обусловливал такое сотрудничество, но не предполагал определение деталей, связанных с областью действия и графиком в рамках выделенного бюджета.

Реакция на изменения и соблюдение плана

Очень часто успех или неудача программного проекта определяется способностью реагировать на изменения. В процессе формирования планов следует удостовериться в том, что эти документы обладают достаточной гибкостью и могут адаптироваться в соответствии с изменениями, вносимыми в процесс работы и технологию.

Ход выполнения программного проекта невозможно запланировать “с прицелом на будущее”. Это связано с тем, что непрерывно изменяется деловое окружение, в результате чего изменяются и требования. Во-вторых, заказчики могут изменить требования непосредственно после начала эксплуатации системы. И наконец, даже если требования известны, и мы уверены в том, что они не изменятся в дальнейшем, вряд ли удастся точно определить время, требуемое на их формулирование.

Обычно начинающие менеджеры увлечены идеями создания PERT-диаграмм или диаграмм Гантта в проекте в целом, а затем их использованием в качестве наглядного пособия. Создается ошибочное впечатление, что подобная диаграмма может контролировать ход выполнения проекта. Появляется возможность отслеживать индивидуальные задания и по факту их выполнения вычеркивать их из диаграммы. Можно сравнивать реальные данные с тем, что запланировано, и реагировать на обнаруженные расхождения.

Но *на самом деле* это приводит к разрушению структуры самой диаграммы. По мере приобретения командой знаний о системе, а клиентами — знаний о своих потребностях, некоторые задачи, предусмотренные диаграммой, становятся ненужными. При этом обнаруживаются новые задачи, которые необходимо внести в диаграмму. Вообще говоря, изменятся не только запланированные данные, но и сама *форма*.

Более эффективная стратегия планирования заключается в составлении детальных планов на последующие две недели, очень приблизительных планов на последующие три месяца и крайне обобщенных планов на более длительные сроки. Требуется заранее знать, какие задачи нам предстоит выполнить на протяжении последующих двух недель. Можно лишь приблизительно знать требования, над которыми мы будем работать последующие три месяца. И у нас должно быть лишь обобщенное представление о том, какие функциональные возможности должна иметь система через год.

Подобное распределение плана по убывающей шкале означает, что мы акцентируем внимание на разработке детального плана, учитывающего выполнение только срочных задач. Как только составлен детальный план, его тяжело изменить, поскольку команда обладает известным "моментом инерции" и некоторой долей неповоротливости. Однако поскольку такой план рассчитан только на четырехнедельный период, остальная часть плана становится более "гибкой".

Принципы быстрой разработки ПО

Вышеприведенные положения послужили причиной появления 12 принципов быстрой разработки ПО, заменяющих традиционный громоздкий процесс разработки.

- *Наша первостепенная задача — достичь соответствия требованиям заказчика, непрерывно поставляя ему в оптимальные сроки высококачественное ПО*. Журнал "MITSloan Management Review" опубликовал анализ правил разработки ПО, которые помогают компаниям разрабатывать продукты высокого качества[3]. Одно из правил акцентировано на взаимосвязи между качеством и поставкой частично функционирующей системы на раннем этапе разработки. В статье сообщалось, что чем *меньшими функциональными возможностями обладает программный продукт на раннем этапе разработки, тем выше качество окончательной версии*. Следующее положение, рассматриваемое в статье, говорит о наличии тесной взаимосвязи между качеством конечного продукта и частыми поставками отдельных модулей с растущими функциональными возможностями. *Чем чаще происходит постав-*

[3] *Product-Development Practices That Work: How Internet Companies Build Software*, MIT Sloan Management Review, Winter 2001, Reprint number 4226.

ка частей программного продукта, чем выше качество конечного продукта. Совокупность методик быстрой разработки ПО предусматривает доставку разработанного компонента проекта на раннем этапе разработки и последующие частые поставки модулей по мере выполнения проекта. Наша цель — поставить элементарную систему на протяжении первых нескольких недель после начала выполнения проекта. Затем каждые две недели мы стремимся поставлять системные модули с растущими функциональными возможностями. Заказчики могут принять решение начать эксплуатацию системных модулей, если они считают, что они обладают достаточным уровнем функциональных возможностей. Или же они могут решить просто пересмотреть имеющиеся функциональные возможности и сообщить разработчикам об изменениях, которые необходимо внести в программный продукт.

- *Следует приветствовать изменения требований, даже если они происходят на позднем этапе разработки. Правила быстрой разработки ПО рассматривают изменения как преимущество в конкурентной борьбе*. Этот принцип отражает сложившуюся систему отношений. Участники процесса быстрой разработки ПО не опасаются изменений. Они рассматривают модификацию требований как *положительный момент*, поскольку при этом предполагается, что команда получила дополнительную информацию о том, каким образом следует соответствовать требованиям потребительского рынка. Команда, участвующая в быстрой разработке ПО, усердно и настойчиво трудится над сохранением гибкости структуры ПО. Благодаря этому в случае изменения требований влияние на систему будет минимальным. В последующих главах этой книги вы ознакомитесь с принципами и шаблонами объектно-ориентированного проектирования, которые позволят сохранять гибкость подобного рода.
- *Следует часто поставлять рабочее ПО: от одного раза в две недели до одного раза в два месяца, стремясь к соблюдению более сжатых сроков*. Команда разработчиков должна поставлять рабочий программный продукт, причем в максимально сжатые сроки (по истечению нескольких первых недель) и часто (каждые последующие несколько недель). При этом не следует довольствоваться поставками пачек документов или планов. Наше внимание сосредоточено на задаче поставить ПО, соответствующее требованиям заказчика.
- *Бизнесмены и разработчики обязаны ежедневно совместно трудиться над проектом на протяжении всего процесса его выполнения*. Чтобы обеспечить быстрое выполнение проекта, между заказчиками, разработчиками и организаторами проекта следует установить тесное и постоянное взаимодействие. Программный проект не должен соответствовать принципу “сделал и забыл”, поэтому проект должен находиться под постоянным руководством.

- *Выполняйте проекты с привлечением мотивированных индивидов. Предоставьте им соответствующую среду разработки и необходимую поддержку, причем доверьте им выполнение поставленных задач*. В проекте быстрой разработки ПО люди рассматриваются в качестве важнейшего фактора успеха. Все другие факторы — процесс, среда разработки, менеджмент и прочие — считаются второстепенными и могут изменяться, если они оказывают на людей неблагоприятное воздействие. Например, если среда разработки в офисе препятствует работе команды, ее следует изменить. Если определенные этапы процесса препятствуют работе команде, их следует изменить.
- *Наиболее результативный и эффективный метод передачи информации команде, а также в рамках самой команды заключается в интерактивном общении*. В проекте быстрой разработки ПО люди общаются друг с другом. Основной метод коммуникации — это общение. Документы можно создавать, но невозможно охватить всю информацию в письменном виде. Команде, работающей над проектом быстрой разработки ПО, не требуются спецификации, планы и проекты, выраженные в письменной форме. Эти документы могут быть сформированы, если возникнет безотлагательная и существенная потребность в этом, но они не являются абсолютной необходимостью. Главное — это общение.
- *Рабочее программное обеспечение — это первостепенный критерий процесса*. Прогресс в проектах быстрой разработки ПО отслеживается путем оценки части ПО, которая в текущий момент времени удовлетворяет требования заказчика. Прогресс подобных проектов не связан с текущим этапом их выполнения, а также не может оцениваться на основе объема выпущенной документации или кода инфраструктуры, разработанного в результате их выполнения. Программный проект считается выполненным на 30%, если получен на 30% функционирующий программный продукт.
- *Проекты быстрой разработки ПО способствуют непрерывному развитию. Спонсоры, разработчики и пользователи должны уметь поддерживать постоянный темп работы*. Проект быстрой разработки ПО — это не спринтерский забег, а марафон. Команда не срывается с места на полной скорости и не пытается удержать эту скорость на протяжении всего проекта. Вместо этого она продвигается быстрым, но равномерным шагом. Слишком быстрое продвижение ведет к истощению сил, появлению недостатков и, в конечном счете, — к неудаче. Команды по быстрой разработке ПО сами поддерживают свой темп. Они не позволяют себе слишком устать. Они не заимствуют частичку завтрашней энергии для того, чтобы успеть сделать чуть больше сегодня. Они работают на скорости, позволяющей поддерживать высочайшие стандарты качества на протяжении всего проекта.

- *Постоянный акцент на высоком техническом уровне качества и хорошем проекте повышает быстроту разработки*. Высокое качество — это ключ к достижению высокой скорости разработки. Быстрого продвижения можно достичь путем поддержки максимального уровня качества и прочности программного продукта. Следовательно, все члены команды по быстрой разработке ПО обязаны создавать код с качественными характеристиками высочайшего качества. Следует избегать формирования "неразберихи", обещая устранить ее при первой же возможности. Если происходит какая-либо путаница, устраняйте ее до конца текущего рабочего дня.
- *Необходимость в простоте*. Команды по быстрой разработке ПО не пытаются построить грандиозную суперсистему. Вместо этого они всегда избирают простейший путь, который согласуется с поставленными целями. Они не придают большого значения прогнозированию завтрашних проблем и не пытаются оградить себя от них уже сегодня. Вместо этого сегодня они делают самую простую и высококачественную работу, будучи уверенными в том, что им удастся легко внести необходимые изменения, если в будущем возникнут какие-либо проблемы.
- *Наилучшие архитектуры, требования и проекты возникают у самоорганизующихся команд*. Команда по быстрой разработке ПО является самоорганизующейся. Обязанности не передаются отдельным членам команды, а учитывается лишь команда в целом. Лишь сама команда определяет наилучший способ их реализации. Члены команды по быстрой разработке ПО совместно работают над всеми аспектами проекта. Каждый из них имеет право внести свой вклад в общее дело. Ни один из членов команды не несет ответственность за архитектуру, требования или тесты. Команда разделяет эти обязанности, и каждый член команды может влиять на этот процесс.
- *Периодически команда делает выводы об эффективности своей работы и соответственно модифицирует и адаптирует свою линию поведения*. Команда по быстрой разработке ПО непрерывно подстраивает собственную организацию, правила, обычаи, взаимосвязи и т.д. Команда по быстрой разработке ПО осознает, что среда разработки постоянно изменяется и что она сама должна меняться вместе со средой, чтобы сохранить быстрый темп разработки.

Резюме

Профессиональная цель каждого разработчика ПО и команды в целом заключается в поставке своим работодателям и заказчикам продукта высочайшего качества. И все же, в удручающем числе случаев наши проекты терпят неудачу, или в результате их выполнения не удается произвести качественный продукт. Да-

же будучи хорошо спланированной, восходящая спираль “раздувания” процесса “виновна”, по меньшей мере, в некоторых из таких неудач. Принципы и критерии быстрой разработки ПО были сформированы с целью помочь команде разорвать замкнутый круг “раздувания” процесса и сфокусироваться на простых средствах для достижения поставленных целей.

На время написания этой книги уже существовало большое количество процессов быстрой разработки ПО. Сюда входят такие процессы, как SCRUM[4], Crystal[5], процесс разработки, управляемый свойствами (Feature Driven Development)[6], адаптивная разработка программного обеспечения (Adaptive Software Development (ADP))[7] и, самое важное, экстремальное программирование (Extreme Programming)[8].

Литература

1. Beck K. *Extreme Programming Explained: Embracing Change*. Reading, MA: Addison-Wesley, 1999.
2. Newkirk J. and Martin R. *Extreme Programming in Practice*. Upper Saddle River, NJ: Addison-Wesley, 2001.
3. Highsmith J. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. NY: Dorset House, 2000.

[4] Web-узел `www.controlchaos.com`.

[5] Web-узел `crystalmethodologies.org`.

[6] Java Modeling In Color With UML: Enterprise Components and Process, Peter Goad, Eric Lefebre, and Jeff De Luca, Prentice Hall, 1999.

[7] [Highsmith2000].

[8] [Beck1999].[Newkirk2001].