IBM PRESS

# СТРАТЕГИЯ УПРАВЛЕНИЯ КОНФИГУРАЦИЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ИСПОЛЬЗОВАНИЕМ IBM RATIONAL CLEARCASE

Практическое руководство

Второе издание

Дэвид Белладжио
Том Миллиган

ДМК ПРЕСС
издательство

**УДК 004.4**
**ББК 32.973.26-018.2**
**Б43**

Б43 **Дэвид Белладжио, Том Миллиган**
Стратегия управления конфигурацией программного обеспечения с использованием IBM Rational ClearCase: Пер. с англ. Мухина Н. – М.: ДМК Пресс, 2007. – 384 с.: ил.

**ISBN 5-94074-358-7**

Управление конфигурацией программного обеспечения (SCM) помогает группам разработчиков справляться с наиболее крупными и сложными проектами для выпуска более качественных программных продуктов.
Второе издание книги описывает последние новшества ClearCase и ClearQuest®, при этом еще глубже проникая в стратегию и управление SCM. Каждый из авторов обладает более чем 15-летним опытом в области SCM и знаниями технологий IBM Rational, применяемых клиентами по всему миру. В этой книге они систематически раскрывают планирование, развертывание, а также применение SCM на протяжении всего жизненного цикла проектов: разработка, интеграция, построение, стабилизация, развертывание версий, и т. д. Они представляют практическое руководство по решению проблем, связанных с ростом и усложнением проектов, - от управления географически распределенными командами разработчиков до отслеживания запросов на изменения.

УДК 004.4
ББК 32.973.26-018.2



ISBN 0-321-20019-5 (англ.)
ISBN 5-94074-358-7 (рус.)

# Содержание

## Глава 2

## Глава 3



## Глава 4

## Глава 5

## Глава 7

## Глава 8



## Глава 9

## Глава 10



## Глава 11

## Глава 12



## Приложение А

# Глава 1

# Что такое управление конфигурацией программного обеспечения?

Заголовок этой главы задает простой вопрос, ответ на который, как многие могут подумать, должен быть известен любому, кто имеет хоть какой-нибудь опыт в разработке программного обеспечения. На самом деле похоже, что очень немногие в состоянии четко сформулировать, что означает термин *управление конфигурацией программного обеспечения*.

Точнее говоря, можно сказать, что те, кто обладает опытом разработки программного обеспечения, знают, что существует необходимость контролировать то, что происходит в процессе разработки, а если уж этот процесс контролируется, то он может быть измерен и направлен. Из осознания такой необходимости происходит четкое рабочее определение управления конфигурацией ПО:

> Управление конфигурацией программного обеспечения – это способ контроля эволюции программного проекта.

Если выразить это более формально, *управление конфигурацией ПО* (Software Configuration Management – SCM) – это дисциплина программной инженерии, включающая средства и методы (процессы или методологию), которые использует компания для управления изменениями в ее программном продукте. Введение к стандарту IEEE «Standard for Software Configuration Management Plans» [IEEE 828-1998] сообщает об SCM следующее:

> SCM представляет хорошую инженерную практику для всех программных проектов, независимо от их типа (поэтапная разработка, быстрое прототипирование или текущее сопровождение). Эта технология повышает надежность и качество программного обеспечения за счет:

- представления структуры для идентификации и контроля документации, кода, интерфейсов и баз данных для поддержки всех фаз жизненного цикла;

- поддержки выбранной методологии разработки/сопровождения, которая отвечает требованиям, стандартам, политикам, организации и философии менеджмента;
- формирования организационной информации и информации о продукте, касающейся состояния базовых линий, контроля изменений, тестов, версий, аудита и т.п.

Понятно, что программное обеспечение легко изменить, даже слишком легко. И мало того, что его легко изменить, но вдобавок эти изменения не ограничены физическими законами, которые в определенной степени служат «оградой» для изменения аппаратных систем. Программное обеспечение ограничено лишь пределами человеческого воображения. Неконтролируемое и неуправляемое воображение быстро приводит к кошмарам.

Сегодня большинство команд разработчиков программных проектов понимают необходимость в SCM для управления изменениями их программных систем. Однако, даже несмотря на наилучшие намерения, программные проекты продолжают терпеть неудачи по причинам, которых можно избежать, применяя инструменты SCM и соответствующие процессы. Эти неудачи выражаются в низком качестве, задержках поставок, перерасходах смет и несоответствии потребностям пользователей.

Чтобы понять управление конфигурацией программного обеспечения, возможно, легче сначала разобраться с управлением конфигурацией аппаратных сред. Аппаратные системы имеют физические характеристики, которые позволяют яснее увидеть последствия недостаточного управления конфигурацией.

Например, рассмотрим персональный компьютер. У компьютера есть процессор, материнская плата, память, жесткий диск, монитор и клавиатура. Каждый из этих аппаратных элементов обладает интерфейсом, соединяющим его с другими элементами. У мышки есть разъем, а у компьютера – порт, в который можно подключить мышку, – и... вуаля! Все работает.

Если разъем мышки окажется несовместим с портом компьютера, то соединить эти две аппаратные части в единую работающую систему будет невозможно. У компьютера много подобных интерфейсов. Процессор и память вставляются в материнскую плату, жесткий диск подключается к контроллеру, принтер, монитор и клавиатура – все они обладают собственными интерфейсами.

Когда производится аппаратное обеспечение, легко увидеть интерфейсы, которые важны для работы готовой системы. Таким образом, они хорошо известны и тщательно учитываются при любых изменениях, вносимых в аппаратный дизайн.

Для аппаратных систем управление конфигурацией имеет следующие аспекты. Каждая система пронумерована, идентифицирована и имеет номер версии. Каждый номер версии указывает различные исполнения одной и той же части. Например, год выпуска модели автомобиля – это номер версии этого автомобиля. Это может быть Honda CRV 2003, Honda CRV 2004 и т. д. Когда изменяется дизайн аппаратной системы, эта система получает новый номер версии.

Аппаратная система может состоять из сотен, тысяч или десятков тысяч частей. Следующее, что необходимо знать, – какие именно версии частей собираются

вместе. В производстве это называется ведомостью материалов, или спецификацией. Эта спецификация перечисляет все составные части и указывает, какие именно версии этих составных частей должны быть использованы для построения системы.

Части собираются в узлы, что упрощает процесс производства крупных систем. В примере с персональным компьютером вы можете сказать, какие версии мышки, жесткого диска, процессора и монитора должны быть собраны вместе, чтобы получилась готовая система. Каждая из этих частей, такая как жесткий диск, состоит из многих, многих подчастей, которые должны быть собраны вместе, чтобы получился работоспособный узел.

Управление конфигурацией программного обеспечения имеет дело с теми же проблемами, что и управление конфигурацией аппаратных систем (и со многими дополнительными, причиной которых служит отсутствие ограничений законов физики). Каждая программная часть имеет интерфейс, и эти «части» стыкуются вместе, формируя программную систему. Части программного обеспечения имеют различные наименования, – такие, как подсистема, модуль или компонент. Они должны идентифицироваться и иметь номер версии. Они также должны иметь совместимые интерфейсы, но разные версии частей могут иметь разные интерфейсы. В конечном итоге вам нужна ведомость материалов, чтобы видеть, какие версии каких компонентов составляют полную программную систему.

Однако правильно управлять конфигурацией программного обеспечения намного сложнее, потому что программы легче изменять, чем аппаратное обеспечение. Несколько нажатий клавиш, щелчок мышки по кнопке Save – и вы создали новую версию программы. В отличие от аппаратной части производство программ – очень быстрый процесс, который может выполняться сотни раз в день каждым участником команды разработчиков. Это то, что обычно называется «выпуском сборки ПО» (software build) или просто «сборкой ПО».

В этой динамичной, изменчивой среде дисциплина SCM призвана обеспечить, чтобы, когда выпускается окончательная версия полной программной системы, все ее составные части были собраны в одно и то же время, в одном и том же месте, чтобы можно было соединить их вместе и заставить работать, как требуется. Хотя большинство групп разработчиков программных проектов понимают необходимость в SCM, многие не могут использовать его правильно не только из-за сложности SCM, но также из-за того, что у них нет четкого понимания того, что конкретно должна делать хорошая система SCM. Чтобы заложить основы правильного понимания, остальная часть этой главы посвящена подробному описанию основных приемов применения SCM, представляет концепции инструментов SCM и процессы, используемые для реализации этих приемов.

## 1.1 Практика применения SCM

Внедряя инструменты и процессы SCM, вы должны определить, какие приемы и политику следует использовать, чтобы избежать распространенных проблем с конфигурацией и максимизировать производительность работы команды.

Многолетний практический опыт показал, что следующие приемы важны для организации успешной разработки программного обеспечения:

- идентификация и хранение рабочих продуктов в защищенном репозитории;
- контроль и аудит изменений рабочих продуктов;
- организация версий рабочих продуктов в версии компонентов;
- организация версий компонентов и подсистем в версии подсистем;
- создание базовых линий в контрольных точках проекта;
- запись и отслеживание запросов на изменение;
- организация и интеграция согласованных наборов версий посредством видов деятельности (activities);
- сопровождение стабильных и согласованных рабочих пространств;
- поддержка параллельных изменений в рабочих продуктах и компонентах;
- раннее начало интеграции системы и частое ее повторение;
- обеспечение воспроизводимости сборок (builds) программного обеспечения.

Далее в этом разделе будет объяснен каждый из этих приемов.

### 1.1.1 Идентификация и хранение рабочих продуктов в защищенном репозитории

Чтобы осуществлять управление конфигурацией, вы должны идентифицировать рабочие продукты, которые должны быть помещены под контроль версий. Эти рабочие продукты должны включать как те, что применяются для управления и проектирования системы (такие как проектные планы и модели проектирования), так и те, что собственно составляют ее проект (такие как файлы исходного текста, библиотеки, исполнимые программы и механизмы, необходимые для их построения). IEEE называет это *идентификацией конфигурации*: «элемент управления конфигурацией, состоящий из выбора конфигурируемых элементов системы и записи их функциональных и физических характеристик в технической документации» [IEEE Glossary, 1990].

В терминах инструмента SCM идентификация означает способность простого и быстрого нахождения и идентификации любого рабочего продукта или системы. Любой, кто управлял разработкой или разрабатывал проект без применения SCM, или с плохим SCM, может подтвердить трудность нахождения «правильной» версии «правильного» файла, когда их копии постоянно перемещаются с места на место. В конечном итоге потеря или неправильная идентификация версий рабочего продукта могут привести к провалу проекта – либо по причине задержки поставки системы из-за потери ее частей, либо по причине снижения качества системы из-за неправильных ее частей.

Но организации рабочих продуктов и обеспечения возможности их быстрого нахождения недостаточно. Вам необходимы устойчивые, масштабируемые, распределяемые и реплицируемые репозитории для всех наиболее важных рабочих продуктов. По мере роста вашей организации вы будете добавлять данные и новые репозитории, поэтому масштабируемость и распределенность необходимы для обеспечения высокой производительности системы.

Другой причиной роста компаний на современном рынке программного обеспечения является приобретение иных компаний, что приводит к географической удаленности работающих совместно групп разработчиков. Поэтому инструмент SCM должен обеспечивать возможность поддержки совместной работы команд, находящихся в различных географически удаленных друг от друга местах.

И наконец, репозитории должны подвергаться резервному копированию и аварийному восстановлению посредством простых процедур. К сожалению, многие компании не уделяют этому должного внимания, что ведет к серьезным проблемам.

### 1.1.2 Контроль и аудит изменений рабочих продуктов

После того как рабочие продукты идентифицированы и помещены в репозиторий, вы должны быть готовы контролировать, кому разрешено модифицировать их, а также сохранять записи о том, в чем именно заключались проведенные модификации, кто их выполнил, когда это было сделано и почему. Мы называем это информацией аудита. Этот прием связан с темой управления конфигурацией, называемой в IEEE как *контроль конфигурации* и *учет статуса конфигурации* и определенной соответственно как «оценка, координация, утверждение либо отказ в утверждении и реализация изменений конфигурируемых элементов» и «протоколирование и отчетность, необходимые для эффективного управления конфигурацией» [IEEE Glossary, 1990].

Применяя приемы контроля и аудита, организация может самостоятельно определять, насколько строгими должны быть ее политики контроля изменений. Без контроля любой может внести изменения в систему. Без аудита вы никогда не будете знать, что в действительности происходит с системой. Имея информацию аудита, даже без ограничения изменений, вы всегда можете видеть, что было изменено, кем и почему. Информация аудита также позволяет вам легко вносить исправления при появлении ошибок. Сбалансированное применение контроля и аудита позволит вам настроить собственный подход к управлению изменениями, наилучшим образом подходящий для нужд вашей организации. В идеале вы должны стремиться к обеспечению максимальной производительности разработки при исключении известных рисков нарушения безопасности.

### 1.1.3 Организация версий рабочих продуктов в версии компонентов

Когда в системе присутствует более чем несколько сотен файлов и директориев, возникает необходимость группировать эти файлы и директории в некоторые более крупные объекты, чтобы упростить управление и избежать организационных проблем. Эти единые объекты, состоящие из наборов файлов и директориев, в программной индустрии имеют много разных наименований, включая пакеты, модули и компоненты разработки. В нашей книге мы будем называть их *компонентами SCM* и определим, как набор взаимосвязанных файлов и директориев, которые снабжаются номером версии, могут быть совместно использованы, построены и указаны в базовой линии как единое целое.

Чтобы реализовать компонентный подход в SCM, вы организуете файлы и директории в единый компонент SCM, который физически реализует логический компонент проекта системы. Технология Rational Unified Process (RUP) называет компонент SCM *компонентной подсистемой* [RUP 5.5, 1999] (RUP – это процесс создания ПО, разработанный и поставляемый компанией Rational Software).

Компонентный подход к SCM представляет множество преимуществ, включая следующие:

- *Компоненты снижают сложность.* Использование более высокого уровня абстракции снижает сложность и делает любую проблему лучше управляемой. Применяя компоненты, вы можете рассматривать лишь 6 сущностей, составляющих проект, вместо 5000 файлов, из которых они состоят. Производя систему, вам нужно будет определить лишь 6 базовых линий – по одной для каждого компонента, вместо того чтобы иметь дело с 5000 версиями 5000 файлов. Легче собирать согласованные системы на основе согласованных базовых линий компонентов, чем из индивидуальных версий файлов. Несогласованность ведет к излишним сборкам и ошибкам, которые обнаруживаются на поздних этапах цикла разработки.
- *Проще идентифицировать уровень качества определенной базовой линии компонента, чем сделать это для многочисленных отдельных файлов.* Базовая линия компонента идентифицирует только одну версию каждого файла и директории, составляющих компонент. Поскольку базовая линия компонента состоит из согласованного набора версий, то компонент может быть собран и протестирован как единый узел. Затем можно пометить уровень тестирования, выполненного на каждой базовой линии этого компонента. Этот метод совершенствует взаимодействия и уменьшает число ошибок, когда над компонентом работают совместно две или более групп разработчиков проекта. Например, одна группа производит компонент базы данных, а другая использует его как часть приложения конечного пользователя. Если группа, занимающаяся приложением, сможет легко находить новейшую базовую линию компонента базы данных, которая прошла интеграционное тестирование, то снижается вероятность того, что она использует дефектный набор файлов.
- *Создание экземпляра физического компонента с помощью инструментального средства позволяет обеспечить его совместное и повторное использование.* После создания базовых линий компонента и идентификации уровня их качества группы разработчиков проекта могут искать разные базовые линии данного компонента и выбирать те из них, которые повторно используются от проекта к проекту. Совместное и повторное использование компонентов практически невозможно, если вы не можете определить, из каких версий каких файлов состоит компонент. Совместное использование компонента разными проектами не практично, если вы не можете определить уровень качества и стабильности каждой его базовой линии.

- *Отображение компонентов логического проекта на компоненты SCM помогает сохранить целостность программной архитектуры.* В итерационном процессе разработки части программной архитектуры строятся и тестируются на ранних этапах жизненного цикла программного обеспечения, чтобы исключить риск. Отображая логические архитектурные компоненты на физические компоненты SCM, вы получаете возможность построения и тестирования отдельных частей архитектуры. Это отображение архитектуры на ее реализацию позволяет получить высококачественный код и более ясные интерфейсы между компонентами за счет сохранения целостности исходной архитектуры инструментом SCM.

### 1.1.4 Организация версий компонентов и подсистем в новые версии подсистем

Чтобы обеспечить управление программными системами высокой сложности, вы должны иметь возможность выйти за пределы управления отдельными компонентами и группировать их в подсистемы. Помимо этого, вы должны также иметь возможность включать в определение подсистем другие подсистемы. Такое рекурсивное определение подсистемы как набора совместимых компонентов и подсистем позволяет иерархически определять невероятно сложные системы, управлять ими и контролировать. Если вернуться назад, к примеру персонального компьютера, то вспомним, что PC состоит из процессора, материнской платы, памяти, жесткого диска, монитора и клавиатуры. Каждая из этих сущностей может быть описана как подсистема, то есть совокупность компонентов и других подсистем. Используя такой рекурсивный метод описания системы, структура всего PC может быть специфицирована и управляема иерархически, нисходя до отдельных частей. Заметьте, что при желании вы можете использовать этот рекурсивный метод, чтобы специфицировать PC вплоть до отдельных атомов. Мало того, подсистемы, определенные в этой иерархии, такие как жесткий диск, могут быть повторно использованы в дизайне других PC. Рисунок 1.1 иллюстрирует иерархическую природу подсистем.

Способность использования системы SCM для рекурсивного определения и управления подсистемами, составляющими программную систему, позволяет вам определять и контролировать очень сложные процессы разработки, выделяя разработку подсистем в отдельные проекты, которые независимо контролируются, управляются и реализуются и которые могут стать кандидатами на повторное использование в других проектах.

### 1.1.5 Создание базовых линий в контрольных точках проекта

В основных контрольных точках (milestones) проекта все рабочие продукты должны быть собраны в единую базовую линию. Другими словами, вы должны записывать версии всех рабочих продуктов и компонентов, составляющих систему или подсистему в определенные моменты времени работы над проектом. Как минимум рабочие продукты должны собираться в базовые линии в каждой основной

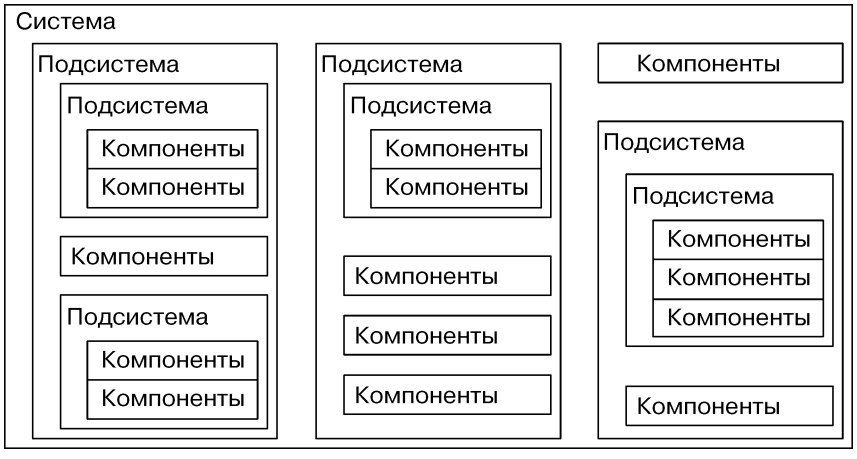


**Рис. 1-1** Декомпозиция крупных систем на иерархически организованные подсистемы упрощает их определение и управляемость

контрольной точке проекта. В итерационном процессе разработки, как это описано в RUP [RUP, 5.5], базовые линии как минимум должны создаваться в конце каждой итерации проекта.

Обычно новые базовые линии создаются намного чаще (иногда ежедневно), ближе к концу итерации или цикла создания версии. Может оказаться полезным создавать базовую линию перед каждой ночной сборкой. Это позволит вам при необходимости воспроизводить любую сборку проекта, запрашивать изменения, внесенные между сборками, и отмечать стабильность сборки в атрибутах качества базовой линии.

Есть три главные причины для создания базовых линий: воспроизводимость, трассируемость и отчетность. Воспроизводимость – это возможность вернуться назад во времени и воспроизвести данную версию программной системы или среды разработки, существовавшей ранее. Трассируемость связывает вместе требования, проектные планы, сценарии тестирования и программные рабочие продукты. Чтобы реализовать ее, вы должны создавать базовые линии не только рабочих продуктов самой системы, но также рабочих продуктов управления проектом. Отчетность позволяет запрашивать содержимое любой базовой линии и сравнивать базовые линии между собой. Такое сравнение может помочь в отладке ошибок и генерации описаний версий.

Хорошая трассируемость, воспроизводимость и отчетность необходимы для решения проблем процесса разработки. Они позволяют вам эффективно исправлять дефекты в поставляемых продуктах, способствовать аудиту ISO-9000 и SEI, и в конечном итоге гарантировать, что проект отвечает требованиям, код реализует проект, и корректный код используется для построения исполнимых программ.

### 1.1.6 Запись и отслеживание запросов на изменение

Управление запросами на изменение программной системы включает отслеживание таких запросов, поступающих извне. Такие запросы могут появляться в результате обнаружения дефектов в процессе тестирования, обнаружения дефектов пользователями, запросов заказчиков на расширение функциональности либо появления новых идей внутри коллектива разработчиков.

Запись и отслеживание запросов на изменение поддерживает контроль конфигурации и изменений, регламентированный IEEE (см. выше в этой главе раздел «Контроль и аудит изменений рабочих продуктов»). Чрезвычайно важно, чтобы запросы на изменения записывались и фиксировался прогресс в реализации или принятии решений о новой реализации. Помимо простого отслеживания запросов на изменение, хорошо организованный процесс управления изменениями позволяет менеджерам проектов выделять приоритеты и планировать даты включения запрошенных изменений в версии готового продукта. Глава 12 описывает эту практику более подробно.

### 1.1.7 Организация и интеграция согласованных наборов версий посредством видов деятельности

Хотя все системы SCM предоставляют возможность контроля версий на уровне файлов, на разработчика возлагается работа по слежению за тем, какие версии каких файлов должны собираться вместе для реализации логически согласованного изменения и обеспечения того, чтобы это изменение было интегрировано как единое целое. Это утомительный ручной процесс, чреватый ошибками. Здесь очень легко допустить ошибку, особенно если разработчик работает более чем с одним изменением одновременно. И это может привести к ошибкам в сборках и потере времени. Такие ошибки проявляются в виде дефектов времени выполнения, которые не воспроизводятся в рабочей среде разработчика.

Некоторые инструменты SCM предоставляют разработчику способ записи запроса на изменение или дефекта, над которыми он работает. Эта информация используется для отслеживания того, какие изменения в файлах и директориях составляют единое логическое изменение. Часто эта информация не используется инструментом SCM, но поддерживается только в целях отчетности и аудита. Ключевая выгода от сбора этой информации об изменении заключается в упрощении интеграционного процесса и обеспечении гарантий согласованности любой конфигурации в любой рабочей среде или среде разработки.

Группирование версий файлов и директориев называется *набором изменения*, или пакетом изменения (в индустрии SCM часто эти два термина различаются. Но разница между ними довольно тонка, и связана с реализацией. Набор изменения определен как действительная «дельта», составляющая собственно изменение, даже если оно измеряется файлами. Пакет изменений – это собранный вместе набор версий файлов. В этой книге мы используем термин из ClearCase – *набор изменения* для обозначения группировки и манипуляций, составляющих изменение). Группировка в основном применяется, когда набор изменений составляет единое логическое изменение. Подход, связанный с применением наборов изме-

нений, применяется уже долгое время. В 1991 г. Питер Филер (Peter Fieler) написал блестящую работу – «Модели управления конфигурацией в коммерческих средах» [Fieler, 1991], описывающую модель наборов изменений.

Набор изменений – это только «клей». Должна быть какая-то связь между изменяемыми версиями и деятельностью, которая является причиной изменения. *Деятельность* (activity) представляет собой единицу выполняемой работы. Деятельности бывают различных типов. Например, дефект, запрос на расширение и его последствия – все это деятельности. Единица работы связывается непосредственно с системой и процессом управления запросами на изменения. Деятельность может быть дочерней по отношению к другой деятельности, которая появляется в системе управления проектом. Способность набора изменений соединять вместе такие дисциплины, как управление конфигурацией, управление запросами на изменение и управление проектом, – наиболее очевидное проявление преимуществ подхода на основе видов деятельности.

Ключевая идея ориентированного на деятельность SCM заключается в повышении уровня абстракции от уровня файлов и версий до уровня видов деятельности. Это достигается прежде всего за счет накопления версий, созданных во время разработки, в единый набор изменений и ассоциации его с деятельностью. Затем эти деятельности представляются повсюду через пользовательский интерфейс и используются операциями SCM для манипуляций согласованными наборами версий.

Преимущества ориентированного на деятельность SCM таковы:

- *Согласованные изменения порождают меньше проблем при сборке и интеграции.* Интеграция согласованного набора изменений (единого логического изменения) уменьшает количество ошибок, происходящих из-за того, что разработчики забывают о некоторых файлах, поставляя свои изменения. Это также гарантирует, что выполняется тестирование согласованных версий – тех же, что были интегрированы, что снижает вероятность ошибок интеграции.
- *Деятельности – это способ логической группировки того, что делают люди.* Обычно разработчики думают о том, над каким средством, запросом на расширение или дефектом они работают. Все они являются типами деятельности. Благодаря такому сконцентрированному на деятельности подходу в инструменте SCM и используя автоматизацию, разработчики могут не задумываться о деталях реализации SCM. Деятельности представляют уровень абстракции, который все участники проекта могут использовать сообща, что позволяет руководителям проектов, тестировщикам, разработчикам, заказчикам и сопровождающему персоналу общаться между собой более эффективно.
- *Деятельности представляют собой естественную связь с управлением запросами на изменение.* Управление запросами на изменение (частью которого является отслеживание дефектов) – важная составная часть работы боль-

шинства организаций, занимающихся разработкой программного обеспечения, а отслеживание выполнения запросов на изменение – одна из ключевых функций SCM. Вместо бессмысленного набора разрозненных версий набор изменений должен быть связан с запросом, зафиксированным в системе управления запросами на изменение. Такая комбинация запросов на изменение с наборами изменений позволяет вести аккуратную отчетность об исправленных дефектах и о том, какие файлы были изменены между базовыми линиями проекта.

- *Деятельности представляют собой естественную связь с управлением проектом.* Менеджеры проектов заинтересованы не только в том, чтобы знать, что было изменено, но также статус изменения, кому оно было поручено и сколько усилий и времени предполагается потратить на его реализацию. Набор изменений связывает данные управления проектом по некоторой деятельности с файлами и версиями, подвергающимися изменению. Эта связь поддерживает автоматизацию, принося выгоды лидеру проекта без потребности в дополнительных усилиях со стороны разработчиков. Например, когда разработчик фиксирует изменение с помощью инструмента SCM, соответствующее изменение статуса деятельности может быть выполнено автоматически и показано в отчете.
- *Деятельности облегчают отчетность.* Основанное на деятельности SCM позволяет всем отчетам и инструментам отображать информацию в терминах проведенных изменений, а не в терминах файлов и их версий, которые составили эти изменения. Это воспринимается более естественно всеми, кто занят в проекте.
- *Деятельности упрощают просмотр кода.* Традиционно, когда выполняется просмотр кода (code review), проверяющий получает список файлов и версий этих файлов, подлежащих проверке. Фокус в том, что нужно знать, с какой версией сравнивать данную, выполняя такой просмотр. Нужно ли сравнивать с непосредственно предыдущей версией, с версией, относящейся к последней базовой линии, или с какой-то другой? Неясно. Имея информацию о наборе изменений, можно заставить инструмент SCM представлять разработчику предыдущую версию автоматически. Это здорово облегчает просмотр кода и делает его менее подверженным ошибкам.
- *Деятельности облегчают тестирование.* Тестирующие организации часто работают со сборками программного обеспечения, «переброшенными через забор» организациями-разработчиками. Им нужно знать, что вошло в сборку или что отличает ее от предыдущей, чтобы решить – что именно следует тестировать и на каком уровне. Большинство тестирующих организаций не обладают ресурсами, необходимыми для прогона полного набора тестов с каждой сборкой, поступившей от разработчиков. Поэтому автоматическая отчетность между двумя базовыми линиями, которая представляет список деятельностей, включенных в последнюю сборку, значительно облегчает работу по сравнению со списком сотен версий файлов, подвергшихся изменениям.

### 1.1.8 Сопровождение стабильных и согласованных рабочих пространств

Разработчику необходимы инструменты и средства автоматизации для создания и сопровождения стабильной рабочей среды. Сопровождение предполагает периодическую синхронизацию изменений с другими членами команды, проводимую таким образом, чтобы в результате получался согласованный набор изменений с известным уровнем стабильности.

Согласованность и стабильность рабочей среды повышает производительность труда разработчика. Без стабильного и согласованного рабочего пространства – закрытых файловых областей, где разработчики могут реализовывать и тестировать код в относительной изоляции, – им пришлось бы тратить значительное время для исследования проблем ошибочных сборок, и иногда они были бы вообще лишены возможности собирать проект в своих собственных пространствах. Проблемы подобного рода могут поглощать время и усилия в любом проекте.

Стабильная модель позволяет изолировать разработчиков от разрушительных изменений, проводимых в других частях проекта, и позволяет им самим принимать решения, когда стоит проводить изменение в общем рабочем пространстве. Стабильная модель также защищает проект от разрушительных и незавершенных изменений, производимых в рабочих пространствах разработчиков.

Согласованная модель предполагает, что когда разработчики обновляют свои рабочие пространства, то эти изменения состоят из известных и проверенных наборов версий, подлежащих сборке.

### 1.1.9 Поддержка параллельных изменений в рабочих продуктах и компонентах

В идеале только один человек должен в единицу времени иметь возможность вносить изменения в любой отдельный файл, и только одна команда разработчиков должна одновременно работать с любым отдельным компонентом. К сожалению, это не всегда эффективно и практично. Самый очевидный случай – это когда необходимо поддерживать одну рабочую версию, в то время как идет разработка следующей.

Ранние инструменты SCM требовали от пользователей сериализации изменений в файлы. Это было неэффективно, поскольку работа одних разработчиков блокировалась и им приходилось ждать, пока другие завершат свои изменения. Это представляло проблему и с точки зрения качества. Блокированные пользователи часто работали над системой, получая копию файла напрямую, в обход процедуры извлечения из репозитория (checking out), и модифицировали его вне контроля SCM. После того как исходный разработчик фиксировал (check in) в репозитории свои изменения, другие получали файл, копировали в него свои изменения и помещали его обратно, таким образом отменяя изменения последней версии файла (см. рис. 1-2).

Эта проблема обычно проявляется в повторном появлении ошибки в поздних сборках системы. Поскольку предполагается, что ошибка исправлена и уже про-

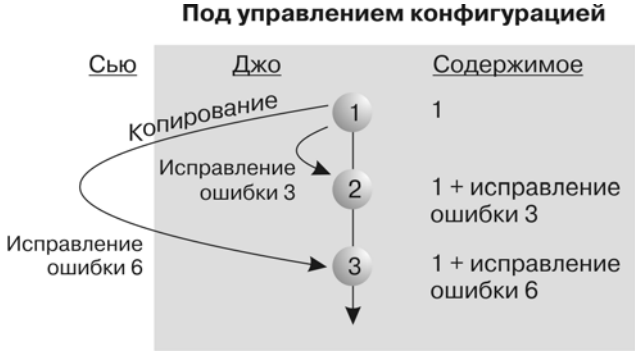


**Рис. 1-2** Проблема последовательной разработки

верена, соответствующее регрессионное тестирование может не выполниться. И если так, то повторно возникшая ошибка попадает в рабочую версию ПО.

Одной из ключевых вещей, которые должен поддерживать инструмент SCM, является возможность параллельной модификации одних и тех же файлов с последующей интеграцией или слиянием этих параллельных изменений. То есть выдвигается требование обеспечения параллельной деятельности в разработке – как упомянутой выше, так и других случаев – например, когда один и тот же разработчик одновременно работает над двумя или более видами деятельности, или же когда несколько разработчиков работают в изоляции над одной функцией, прежде чем она будет введена в проект. Это может означать интегрирацию во время фиксации в репозитории для двух разработчиков, работающих над одним проектом, либо интеграция может быть запланирована для автоматического выполнения на более позднее время, когда объединяются все накопленные изменения. При наличии такой поддержки у пользователей не возникает необходимости «обходить» систему или ожидать снятия блокировки.

### 1.1.10 Раннее начало интеграции системы и частое ее повторение

Во время интеграции обнаруживаются проблемы с интерфейсами и непониманием в проекте. Обе эти проблемы могут оказать значительное влияние на план разработки, поэтому их ранее обнаружение весьма желательно. Планируйте точки интеграции на самых ранних этапах жизненного цикла разработки, и повторяйте ее затем регулярно.

Если у вас принято вести разработку в изоляции (см. «Сопровождение стабильных и согласованных рабочих пространств» ранее в этой главе), то может случиться, что вы установите модель, в которой легко будет оставлять разработчиков чересчур изолированными. Эта проблема может возникнуть независимо от используемого инструмента, поскольку она в большей степени связана с его применением, чем с самим инструментом.