Контроль изменений кода при разработке встраиваемых программных приложений

В статье приводится терминология и принципы управления версиями на примере системы Subversion. Также рассматриваются преимущества и удобство работы с системой контроля версий через графический интерфейс, интегрированный непосредственно в среду разработки программ, на примере взаимодействия системы Subversion с профессиональной средой разработки встраиваемых приложений Atollic TrueSTUDIO.

Анна СЕРГЕЕВА
annserge@rambler.ru

Разработка современных программных приложений связана с постоянными изменениями их исходного кода. Часто в этом процессе задействованы целые команды специалистов. Для отслеживания и упорядочивания всех изменений в коде разрабатываемых программ настоятельно рекомендуется применять специальные методологии и инструменты контроля изменений. Это позволяет предотвратить неизбежную путаницу в ходе реализации проекта, сократить время разработки и в конечном итоге повысить качество программного обеспечения.

Введение

Еще совсем недавно большинство встраиваемых приложений насчитывали лишь несколько модулей исходного кода, созданных одним автором. Однако современные проекты все чаще требуют поддержания порядка сотен модулей. Некоторые из них разрабатываются усилиями участников проекта, другие приобретаются у сторонних поставщиков, поступают от субподрядчиков или копируются из свободных источников с открытым исходным кодом. Так что в настоящее время разработка программных приложений перестает быть просто процессом написания и отладки кода. Теперь это довольно серьезная программная инженерия, которая предъявляет все больше требований к усовершенствованному проектированию, глубокой интеграции и тщательному тестированию [1].

О необходимости управления изменениями в коде

По мере развития проекта исходный код приложения претерпевает тысячи изменений. Рано или поздно инженеры вынуждены возвращаться к более старым версиям разрабатываемого и поддерживаемого ими кода с целью его обновления, исправления обнаруженных ошибок или же для внедрения новых функциональных возможностей.

Каждый, кто сталкивался с этим на практике, знает всю сложность процесса взаимодействия с унаследованным кодом. Часто сопроводительная и проектная документация, которая должна быть предоставлена в виде комментариев к коду, диаграмм и графиков, может иметь формальный характер, быть недостаточной, неактуальной или же вовсе отсутствовать. Таким образом, бывает довольно затруднительно получить ясную и подробную картину хода развития проекта и ориентироваться во всем многообразии внесенных изменений и дополнений в коде. Очень быстро теряется понимание того, кто выполнил те или иные изменения в коде, когда это произошло и для каких целей было сделано.

Возьмем такой распространенный пример. Нередко исправление существующих ошибок в коде способно приводить к непреднамеренному привнесению новых ошибок. Возникает необходимость возвратиться к предыдущему, заведомо работоспособному состоянию кода.

Без применения каких-либо методологий контроля версий вся ценная информация о том, как ранее, до изменений, выглядел оригинальный код, может быть навсегда утрачена с течением времени. Это делает невозможным быстрое восстановление работоспособности кода. В таких ситуациях команды разработчиков теряют много лишнего времени на необоснованные, избыточные повторные действия, которых можно было бы избежать.

Существенную помощь даже в самых сложных и запутанных ситуациях способны оказывать механизмы управления историей изменений разрабатываемого и поддерживаемого кода. Они позволяют определять, кто внес те или иные изменения, когда они были выполнены и для каких целей. Эти весьма важные разъяснения просто необходимы для повышения эффективности обращения с существующим кодом и для внесения всех актуальных изменений своевременно.

На практике применяются достаточно разноплановые подходы к управлению изменениями в коде разрабатываемых программных приложений. Начиная от обновления построчных комментариев в файлах с исходным кодом и отслеживания изменений в электронных таблицах и заканчивая применением целых специализированных программных приложений для отслеживания изменений в коде в процессе его разработки.

Такие приложения называются инструментами контроля версий кода и являются наиболее подходящими и эффективными средствами в решении подобных задач. И хотя на обучение работе с этими инструментами требуется затратить определенное количество времени и усилий, они быстро окупятся за счет преимуществ от применения этих инструментов. В следующих разделах статьи будет описана работа с одной из распространенных в настоящее время систем контроля версий кода — Subversion.

О необходимости управления работой в команде

Использование управления версиями кода позволяет дисциплинировать всех членов команды разработчиков и наладить производственные взаимосвязи между ними, вне зависимости от личных черт характера и каких-либо персональных предпочтений.

Дополнительную сложность вносит и тот факт, что сегодняшя команды разработчиков часто многочисленны и рассредоточены по разным зданиям, городам, странам или даже континентам. Взаимодействие между ними становится проблематичным из-за разницы часовых поясов или деловых поездок, которые затрудняют выяснение таких простых, казалось бы, вопросов, как исправление той или иной ошибки или выполнение за-

прошенных изменений в коде. Применение контроля версий упрощает получение подобной информации даже тогда, когда коллеги по тем или иным причинам не доступны для личного общения.

В командах, применяющих контроль версий, разработчики автоматически взаимодействуют друг с другом каждый раз, когда выполняют наиболее значимые для проекта операции, такие как помещение кода в репозиторий и извлечение из него, формирование и слияние ветвей кода, присвоение тегов и т. д.

Это помогает членам команд разработчиков ориентироваться в существующей структуре сложных проектов, а также дисциплинирует их при документировании всех новых вносимых изменений. Так что в любой момент можно получить достаточно полную информацию о ходе развития проекта.

Об управлении проектами для одиночных разработчиков

Для проектов, реализуемых силами одного разработчика, контроль версий также имеет ряд преимуществ. В большинстве подобных проектов один и тот же специалист может нести ответственность не только за создание программного обеспечения, но и за проектирование аппаратуры, составление спецификаций и сопроводительной документации, заказ и закупку комплектующих и так далее.

Разумеется, в подобных условиях проблема нехватки времени становится катастрофической. Применение контроля версий помогает инженерам оперативно возобновлять работу над текущим проектом после неизбежных перерывов. В долгосрочной перспективе данный подход позволяет предотвращать, например, такие ситуации, когда исполнители забывают исправить важные ошибки и существенные дефекты или же выполняют реинжиниринг старого кода, хотя в этом не было необходимости [2, 3].

Аргументы за и против

Большинство разработчиков уже применяет какие-либо системы контроля версий. Но есть и такие, кто совершенно ими не пользуется. Вот какие аргументы они приводят.

Одни из них сомневаются, что время, затраченное на изучение работы с инструментами контроля версий, успешно окупится. Другие утверждают, что не менее эффективны и не столь строгие методы, такие как регулярное резервное копирование и/или отслеживание изменений в электронных таблицах.

Одиночные разработчики часто утверждают, что поскольку они являются единственными авторами и модификаторами кода, то управление версиями им не нужно, ведь они способны держать всю подобную информацию в своей голове.

Так или иначе, все эти аргументы довольно сомнительны. Большинство организаций, профессионально занимающихся разработкой программных приложений, в числе официальных корпоративных требований к внутреннему распорядку настаивают на обязательном применении специальных инструментов контроля версий [4].

Использование контроля версий также является хорошей практикой в разработке программного обеспечения и обязательным условием в создании критичного программного обеспечения — все это чрезвычайно важно в таких сферах, как управление производственными процессами, авионика или медицинские устройства.

Еще одним фактором, тормозящим применение контроля версий, является то, что внедрение специальных инструментов контроля версий требует приобретения, установки и поддержки отдельного приложения. К тому же необходимо еще и научиться им пользоваться. Безусловно, это так. Однако усилия, затраченные на внедрение разных инструментов, значительно варьируются.

Существует достаточно широкий выбор клиентов для работы с системами контроля версий. Одни действуют из командной строки, другие имеют графический интерфейс. Некоторые среды разработки, хотя и оснащены собственными интегрированными инструментами, поддерживают очень ограниченный интерфейс взаимодействия с системами контроля версий. Другие спроектированы сторонними производителями и недостаточно хорошо интегрированы в саму среду разработки. В некоторых случаях подобные инструменты ограничивают доступ к полному перечню возможностей контроля версий, и их нельзя назвать идеальными. Вот почему так важно внимательно отнестись к выбору конкретного инструмента контроля версий.

От теории систем контроля версий к практике интеграции в профессиональные среды разработки встраиваемых приложений

В данной статье описаны преимущества использования системы контроля версий Subversion в контексте разработки программного обеспечения для встраиваемых систем.

Также показано удобство и практическая эффективность интеграции графических клиентов в систему Subversion на примере такого клиента, предусмотренного в комплекте поставки профессиональной среды для разработки встраиваемых приложений Atollic TrueSTUDIO [5].

Данный клиент с удобным графическим интерфейсом пользователя, интегрированный непосредственно в среду разработки, способен существенно увеличить производительность команды инженеров, поскольку им не придется покидать среду разработки, выполнять рутинные операции по контролю версий в каком-либо отдельном клиенте, а затем снова возвращаться в среду разработки для продолжения проектирования, программирования, отладки и других основных рабочих действий.

Статья состоит из двух взаимосвязанных частей:

- Организация контроля версий разрабатываемого программного обеспечения с использованием системы Subversion.
- Преимущества интегрирования со средой разработки программного обеспечения.

В первой части дан обзор преимуществ использования самих систем контроля версий в процессе разработки программных приложений. Здесь разработчики могут ознакомиться с новыми концепциями современных систем контроля версий, с принципами и механизмами их применения, а также с предназначением конкретных возможностей. В частности, приводится описание принципов работы с системой Subversion. Освоение таких инструментов значительно упрощается, если начинать с понимания базовой модели, на которой строятся инструменты контроля версий, и соответствующих ей терминов. Изложенные основные понятия позволяют значительно сократить время обучения работе с такими системами, как Subversion.

Во второй части внимание акцентировано на подробном обзоре практических преимуществ от глубокого интегрирования графических клиентов системы Subversion непосредственно в среду программной разработки. Рассматривается интеграция с Atollic TrueSTUDIO, одной из наиболее популярных профессиональных сред разработки современных встраиваемых приложений с кодом на C/C++.

Организация контроля версий разрабатываемого ПО с использованием системы Subversion

Subversion — это одна из наиболее широко применяемых систем контроля версий в отрасли разработки программного обеспечения. В основном она получила популярность благодаря своей стабильности, доступности практически на всех платформах и богатому набору поддерживаемых функций.

К тому же Subversion — приложение с открытым исходным кодом, размещенным в открытом доступе [6].

Это еще одна причина ее широкого распространения и применения в отрасли разработки программного обеспечения. Благодаря большому количеству пользователей и поддержке масштабирования Subversion постоянно становится все более стабильной и полезной системой. К тому же она хорошо подходит для разработки критически важных программных проектов силами крупных распределенных команд разработчиков [7].

Система Subversion создана вместо системы CVS, прежде одной из наиболее популярных систем контроля версий в области разра-

ботки программного обеспечения, но теперь заметно устаревшей, а потому нуждающейся в замене более современными аналогами.

Основные принципы работы Subversion

Система контроля версий Subversion управляет файлами и их изменениями с течением времени. Она отслеживает и предоставляет количественные показатели всех изменений во всех файлах на протяжении всего жизненного цикла проекта.

Subversion отслеживает изменения не только в самих файлах, но и в директориях. Кроме того, используемая методология контроля версий позволяет работать не только с текстовыми файлами в кодировке ASCII, такими как файлы заголовков и файлы исходного кода.

Поддерживается хранение с контролем версий для любых типов файлов, в том числе для бинарных, изображений, аудио- или даже офисных документов. Это особенно важный момент, поскольку многие из перечисленных типов файлов имеют немалое значение для поддержания структуры разрабатываемого приложения, а также используются для хранения сопроводительной документации и спецификаций.

Subversion позволяет извлекать старые версии файлов для их повторного применения или изучать изменения между любыми двумя версиями одного и того же файла. Subversion отслеживает все коррективы, внесенные в файл, поэтому можно наблюдать, как он изменился с течением времени. Также можно увидеть, кто внес те или иные изменения, когда и почему это произошло. Если какое-либо конкретное изменение кода привело к проблеме в работе приложения, достаточно просто вернуться к предыдущей стабильной версии, чтобы отменить изменения и продолжить работу [8].

В сущности, системы контроля версий — это своего рода машины времени, которые позволяют перемещаться вперед и назад во времени и видеть, как выглядели рабочие файлы и директории в определенный момент в течение всего жизненного цикла проекта.

Клиент-серверная модель системы

При организации системы хранения версий удобнее всего использовать серверы баз данных. Участники команды разработчиков будут получать доступ к нему через сетевое соединение, с помощью клиент-серверной модели.

Этот подход имеет ряд важных преимуществ. Во-первых, все нужные файлы проекта находятся в централизованном хранилище, доступном для всех разработчиков команды. Во-вторых, в один и тот же момент сразу несколько разработчиков имеют возможность вносить изменения в один и тот же файл, и эти изменения могут быть впоследствии объединены.

В редких случаях, если в одной и той же строке обнаруживаются правки сразу нескольких разработчиков, то специальный менеджер конфликтов тут же обнаруживает эту ситуацию и предоставляет подходящие варианты для урегулирования конфликтующих изменений кода.

Все версии отслеживаемых файлов хранятся в централизованном месте, называемом репозиторием. Доступ к нему в системе Subversion осуществляется с клиентских компьютеров в любой точке сети. Здесь предусмотрено использование как локального (LAN), так и глобального (WAN) подключения. Таким образом, разные разработчики, даже находящиеся в различных географических точках, получают возможность работать с одними и теми же файлами исходного кода.

Следовательно, уже несущественно, находится команда разработчиков в одном месте или рассредоточена по городам, странам или континентам. Все участники проекта обладают равным доступом к общему хранилищу, независимо от своего физического местоположения.

Базовые понятия систем контроля версий

Следующий раздел содержит обзор принципов работы с системой Subversion и охватывает такие важные понятия, как репозиторий Subversion, локальные рабочие копии, проверка версий кода, передача кода в централизованное хранилище и т. д.

Принципы организации репозиториев в системе Subversion

Система Subversion хранит все свои данные, то есть все файлы и историю их изменений, в централизованном месте, называемом репозиторием. Как правило, такой репозиторий находится на сетевом сервере, выделенном под нужды команды разработчиков программного обеспечения.

В соответствии с учетными настройками безопасности, к репозиторию на сервере может подключаться любое количество разработчиков и с помощью клиента Subversion получать доступ к хранящимся на сервере файлам и директориям.

При организации репозиториев соблюдаются следующие основные принципы:

- Репозиторий содержит полное дерево всех файлов и директорий в проекте. В нем имеется главная копия всех файлов исходного кода и возможно вспомогательных файлов и документов.
- При записи файла в репозиторий система Subversion запоминает предыдущее состояние изменяемого файла и только затем уже обновляет его, так что внесенные изменения немедленно становятся доступными для других разработчиков в команде.
- При чтении файла из репозитория предоставляется последняя версия файла, включая все последние изменения, сделанные другими разработчиками.

Существенное различие между Subversion и обычным файловым сервером, который дополнительно хранит файловое дерево, заключается в том, что Subversion запоминает каждое изменение, которое когда-либо было сделано для каждого файла. Система также помнит каждое изменение, связанное и с директориями: создание, переименование или удаление файлов, или просто их перемещение в структуре директорий.

При чтении файлов из репозитория по умолчанию предоставляется последняя версия извлекаемого файла или набора файлов. Кроме того, в клиентах Subversion предусмотрена и возможность получения любой более ранней версии. Это позволяет использовать Subversion для ответа на вопросы «кто сделал последнее изменение этого файла и какие это были изменения?» или «как выглядел этот файл на прошлой неделе?».

Возможность подобного достаточно полного отслеживания версий в репозитории Subversion обеспечивает информацию обо всех когда-либо выполненных изменениях в каждом файле и директории проекта.

Рабочие копии

Рабочая копия в системе Subversion — это файловое дерево из репозитория, скопированное на локальный компьютер. Рабочая копия содержит полный или частичный набор файлов проекта и может рассматриваться как частное, временное рабочее пространство конкретного разработчика.

На рис. 1 представлен пример существования нескольких рабочих копий одного проекта. Например, каждый из разработчиков Joe и William имеет собственную рабочую копию проекта, извлеченную в версии Rev1. Каждый может вносить собственные изменения в исходные файлы и затем снова помещать их в хранилище, независимо друг от друга.

Другие разработчики оперируют с локальными копиями иных версий того же самого проекта: Caroline с версией Rev2, а Mark с версией Rev3.

Поскольку рабочая копия — это личная рабочая зона конкретного специалиста, он может оперировать с исходным кодом по своему усмотрению. Можно редактировать, компилировать, отлаживать, тестировать, вносить новые функциональные возможности и т. д. И пока файлы хранятся только в текущей локальной рабочей копии конкретного разработчика, все изменения в файлах остаются невидимыми для других участников команды. А значит, при работе с локальными копиями действия разных разработчиков в проекте не будут оказывать взаимного влияния.

Каждый разработчик может иметь любое количество рабочих копий, одновременно хранящихся в различных местах на его локальном жестком диске. Эту возможность очень удобно использовать в следующих ситуациях:

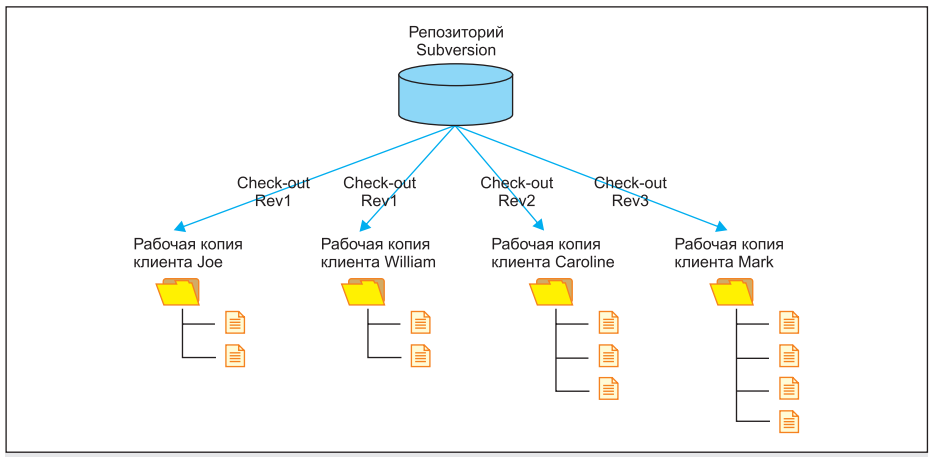

Рис. 1. Рабочие копии в системе Subversion

- Можно работать над реализацией нескольких новых функций или над исправлением нескольких ошибок параллельно. И при этом все правки не окажут никакого взаимного влияния. Так что любые обновления кода могут быть зафиксированы в хранилище по отдельности друг от друга, из разных рабочих копий и в разное время. Subversion гарантирует правильное отслеживание всех изменений, вносимых параллельно за определенный период времени.
- Разработчику, оперирующему в настоящее время с последней версией кода, может понадобиться открыть и более раннюю версию для проведения сравнительного анализа. Может возникнуть необходимость просмотреть предыдущий исходный код или же скомпилировать и запустить его, чтобы изучить поведение кода во время выполнения. В этом случае для локального размещения файлов предыдущей версии из репозитория желательно создать отдельную рабочую копию. Преимущество здесь в том, что манипулирование с файлами прежних версий не будет влиять на работу, выполняемую с файлами из последней версии в текущей рабочей копии.

Другими словами, рабочая копия является изолированной, частной областью, где инженер может работать с исходным кодом определенной версии, не оказывая влияния на код, доступный другим сотрудникам, или на код других версий.

Импортирование

Для создания новых репозиториев в системе Subversion есть специальные административные инструменты. В дальнейшем разработчики могут самостоятельно наполнять его файловое дерево, уже за пределами существующего хранилища. Например, ведущий разработчик в своей локальной рабочей копии создал определенную структуру проекта и желает поместить ее в общедоступное хранилище.

Для импортирования файлового дерева произвольной сложности в центральный репозиторий в системе Subversion предусмотрена специальная операция import (рис. 2). После того как репозиторий будет заполнен начальными файлами и директориями, все участники команды смогут скопировать их в свою локальную рабочую копию и приступить к работе над ними, периодически синхронизируясь с системой контроля версий Subversion.

Извлечение кода из репозитория

Для создания локальной рабочей копии и ее заполнения копией файлов из центрального репозитория выполняется специальная операция извлечения check-out (рис. 3).

В результате выполнения операции в определенном месте на локальном диске создается рабочая копия и в нее помещаются локальные копии файлов, доступные для автономной работы. Любые изменения с ними никак не затрагивают содержимое общедоступного репозитория до тех пор, пока не будет выполнена операция фиксирования обновлений (commit, рассмотрена в следующем разделе).

На рис. 4 показано, что работа с локальной копией может вестись параллельно с централизованным репозиторием и вне зависимости от него.

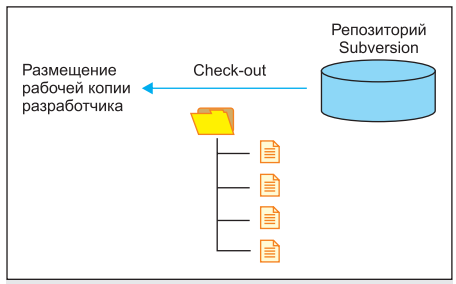

Рис. 3. Извлечение файлов (check-out) из системы Subversion

По усмотрению разработчиков допускается извлечение различных частей файлового дерева репозитория в разные локальные рабочие копии. Также можно получать различные версии файлов или директорий из центрального репозитория и помещать их в различные рабочие копии. Это позволяет иметь на локальном компьютере множество рабочих копий для разных целей, как было описано ранее.

Фиксирование обновлений кода

После внесения в файлы локальной рабочей копии всех изменений, необходимых в данный момент, у разработчика возникает необходимость обновить содержимое репозитория, чтобы сделать свои изменения частью последней «официальной» версии кода (иначе называемой магистральной линией разработки).

Для этого используется операция фиксирования обновлений кода в репозитории (commit), когда содержимое локальной копии обновляет содержимое репозитория. В результате выполнения такой операции содержимое локальной версии помещается в репозиторий Subversion на сетевом сервере, и все другие участники команды получают доступ к данным изменениям (рис. 5).

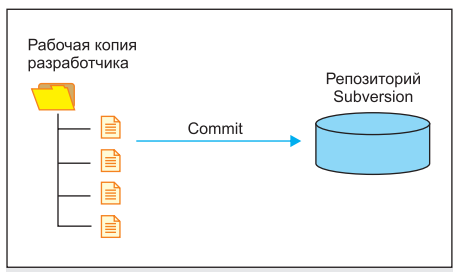

Рис. 5. Фиксирование (commit) обновлений кода в Subversion

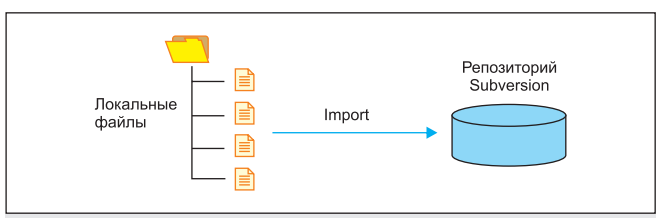

Рис. 2. Импортирование (import) файлового дерева в систему Subversion

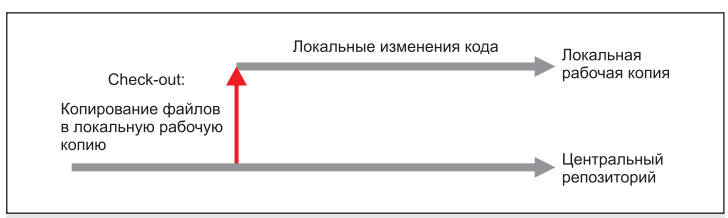

Рис. 4. Раздельное существование локальных и общедоступных копий файлов в Subversion

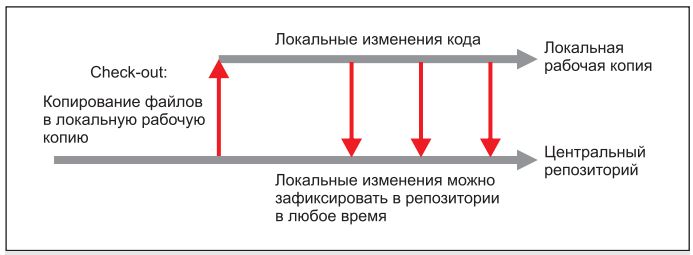

Рис. 6. Фиксирование (commit) обновлений кода в репозиторий возможно в любой момент

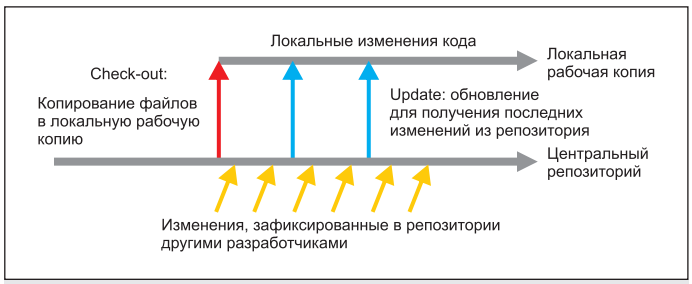

Рис. 7. Обновление (update) кода из репозитория

Локальные изменения автоматически объединяются с последней версией кода в репозитории (рис. 6). По сути, разработчик публикует свои изменения и предоставляет доступ к ним всем другим участникам команды, когда они будут считывать самые последние файлы из репозитория Subversion в свои локальные рабочие копии.

Когда изменения из локальной рабочей копии фиксируются в репозитории, это выполняется в рамках одной атомарной операции. Соответственно, или все изменения записываются в репозиторий единовременно, или же, в случае возникновения проблем, никаких изменений в хранилище записано не будет.

Это чрезвычайно важный момент, предоставляющий гарантию, что не возникнет ситуации, когда в репозиторий попадет неполный набор обновленных файлов, что может вызвать рассогласование версий.

Обновление из репозитория

Допустим, один разработчик достаточно долго работал со своей локальной копией, и за это время другие специалисты успели внести собственные изменения в код и зафиксировать их в центральном репозитории, сделав их общедоступными.

В таком случае при желании разработчик может объединить эти последние изменения из репозитория (совершенные другими разработчиками) с собственной локальной рабочей копией. Для этого используется операция обновления (update). Таким образом, локальная рабочая копия обновляется с учетом актуальных изменений, проделанных другими участниками команды (рис. 7).

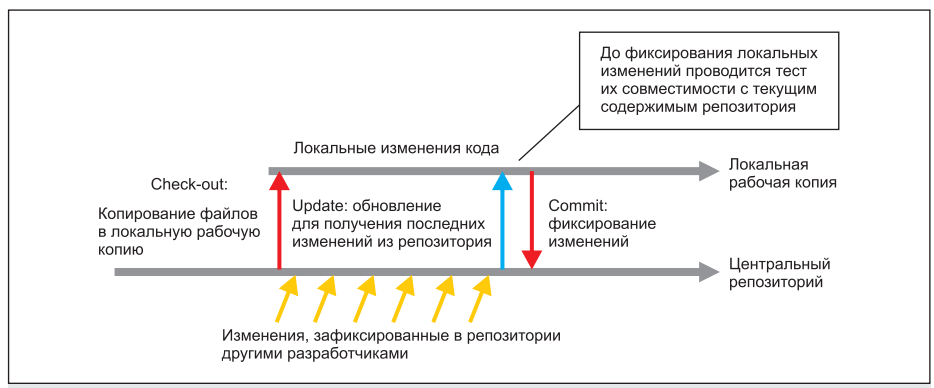

Рис. 8. Обновление (update) кода до фиксации собственных изменений

Вообще, среди разработчиков программного обеспечения считается хорошей практикой до фиксации собственных изменений в общедоступном репозитории выполнять обновление своей рабочей копии с учетом последних изменений из репозитория и обязательно проверять корректность компиляции такого совместного кода (рис. 8).

Если все участники команды применяют этот алгоритм, можно гарантировать, что последние изменения каждого конкретного разработчика не вызовут проблем с изменениями других разработчиков, зафиксированными в репозитории в последнее время.

Ревизии кода

Каждая операция фиксирования изменений (commit) создает новое состояние файлового дерева в репозитории. Каждое такое состояние называется ревизией кода (revision).

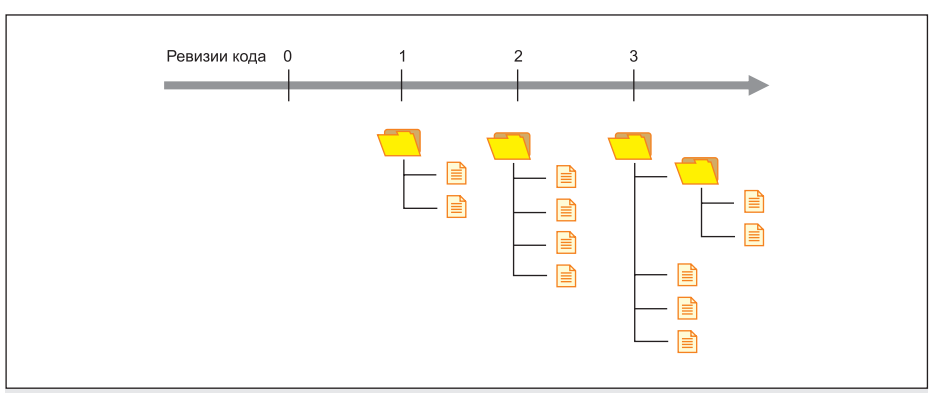

Рис. 9. Ревизии в системе Subversion

Новый, только что созданный репозиторий получает номер ревизии 0 (поскольку на данный момент выполнено еще 0 фиксаций изменений). В дальнейшем, по результатам выполнения каждой операции фиксирования изменений (commit), номер ревизии увеличивается на единицу, отражая таким образом новое состояние файлов в репозитории.

Это можно проиллюстрировать в виде последовательности номеров ревизий, с каждым из которых связано очередное состояние файлового дерева в репозитории (рис. 9).

Здесь ревизия 0 является исходной, в ревизии 1 появляется центральная директория с несколькими файлами, в ревизии 2 набор файлов дополняется, а в ревизии 3 изменяется структура дерева директорий и набор файлов в них.

Ветвление и слияние кода

Как было описано ранее, система хранения и контроля версий Subversion предоставляет множество преимуществ для эффективного взаимодействия участников команды разработчиков, для удобного отслеживания всех вносимых изменений и для возможности быстрого возвращения к прошлым стадиям разработки.

В качестве полезного дополнения Subversion поддерживает возможности ветвления и слияния кода. В данном разделе приводится разъяснение, когда это может быть полезно.

При использовании репозитория Subversion инженер обычно начинает работу в магистральной ветви дерева — именно той ветви, где по умолчанию все изменения кода сохраня-

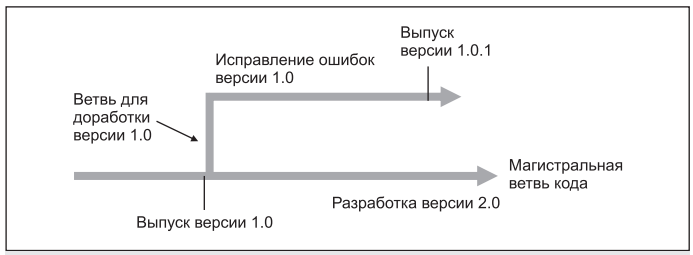

Рис. 10. Ветвление в системе Subversion для выпуска завершенных версий программного продукта

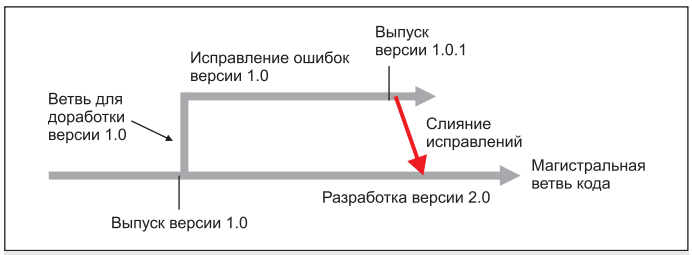

Рис. 11. Слияние кода для исправления ошибок в нескольких версиях программного продукта

ются посредством операции commit и извлекаются с помощью операции check-out.

Это значит, что все изменения будут выполняться в одной и той же линии разработки. И если действовать только по данной схеме, в определенных (вполне распространенных) ситуациях это может привести к существенным проблемам.

На практике часто появляется необходимость в создании параллельных, не зависящих друг от друга линий разработки одного и того же кода. Для этого в Subversion предусмотрены операции ветвления (branch) и слияния (merge). То есть разработка может вестись в нескольких параллельных потоках, которые не будут мешать друг другу.

При создании ветви, по сути, формируется копия состояния магистральной версии кода в конкретный момент времени. Далее разработка может производиться как в магистральной части репозитория, так и в его ветви. При этом любые изменения кода, зафиксированные в магистрали, не будут пересекаться с изменениями, зарегистрированными в ветви, и наоборот.

Когда же возникает потребность копирования содержимого репозитория в параллельные, независимые ветви? Существует два распространенных сценария:

- Создание версионных ветвей (release branches) для выпуска очередных завершенных версий программного продукта.
- Создание функциональных ветвей (feature branches) для внедрения новых функциональных возможностей.

Далее приводится описание преимуществ использования ветвления и слияния в этих двух сценариях.

Ведение версионных ветвей

По данному сценарию создание ветвей используется для работы с программным продуктом, перед командой разработчиков стоит задача выпускать несколько завершенных версий одного продукта параллельно.

Предположим, некоторое время назад уже была выпущена версия какого-либо программного продукта с номером 1.0. И сразу же после этого началась работа над очередной готовящейся к выпуску версией 2.0, в которую будет включен большой набор новых функциональных возможностей. В самом простом случае все эти изменения будут сохраняться в магистральную ветвь репозитория. Однако в то же самое время, пока работы над текущей версией 2.0 все еще продолжаются, вдруг становится известно о критических проблемах, обнаруженных в выпущенной ранее версии 1.0.

В такой ситуации менеджмент проекта сталкивается с дилеммой, поскольку выпуск версии 2.0 планируется только через несколько месяцев, а исправить проблемы в уже выпущенной и предоставленной клиентам версии 1.0 нужно немедленно. Ожидать же завершения работ над текущей версией 2.0 в данном случае нет никакой возможности.

Именно здесь и приходит на помощь ветвление.

При этом делать снимок состояния кода наполовину готовой версии 2.0, вносить в него срочные исправления ошибок, обнаруженных в версии 1.0, и выпускать его в качестве версии 1.1 будет нецелесообразным. Ведь новые функциональные возможности, частично реализованные в версии 2.0, еще не являются стабильными.

Для продолжения работ с обеими версиями, 1 и 2, следует применять другое решение, его суть отображена на рис. 10.

Корректный вариант — вернуться к коду выпущенной версии 1.0, сделать его отдельную копию и внести необходимые исправления только тех ошибок, которые были обнаружены именно в версии 1.0, без добавления каких-либо новых функциональных возможностей, связанных уже с версией 2. Такая исправленная версия 1.0 может быть выпущена, скажем, с номером 1.0.1.

И при этом никакого влияния на ход выполнения работ по готовящейся к выпуску версии 2.0 оказано не будет. Так что работа над кодом 2.0 может продолжаться параллельно с исправлением ошибок более старых версий. Изменения кода, зафиксированные в репозитории для каждого из проектов, не оказывают взаимного влияния, поскольку исправление ошибок версии 1.0 совершается в отдельной ветви, в то время как все изменения версии 2.0 фиксируются в магистральной ветви.

Теперь проследим, что же произойдет с исправлениями, реализованными для версии 1.0.1. Они не должны быть оставлены без внимания, поскольку должны быть тоже включены и в 2.0. Ведь в противном случае код версии 2.0 будет наследовать неисправленные ошибки из версии 1.0 (рис. 11). Здесь на помощь приходит слияние ветвей кода (merging).

Для того чтобы избежать этой ситуации, все изменения кода, выполненные в ветви для выпущенной ранее версии 1.0 и попавшие в исправленную версию 1.0.1, необходимо перенести и в код магистральной ветви, где ведется разработка текущей версии 2.0.

По ходу развития проекта, если в версии 1.0.1 обнаружатся дополнительные ошибки и это повлечет необходимость создания версии 1.0.2, будет совершенно ясно, как справиться и с этим.

Нужно продолжать работать с кодом в ветви версии 1.0.1 и вместе с тем исправлять ошибки для версии 1.0.2, а затем слить эти изменения обратно в основную ветвь кода, так что в конечном итоге они будут включены и в выпуск версии 2.0 (рис. 12).

Для описанного сценария есть еще один показательный вариант, когда от магистральной ветви разработки отделяется новая ветвь для сохранения ключевого состояния кода очередной, предстоящей к выпуску версии. Это именно тот момент, когда этап разработки всех новых функциональных возможностей уже завершен, а этап тестирования и выпуска еще не начат (рис. 13).

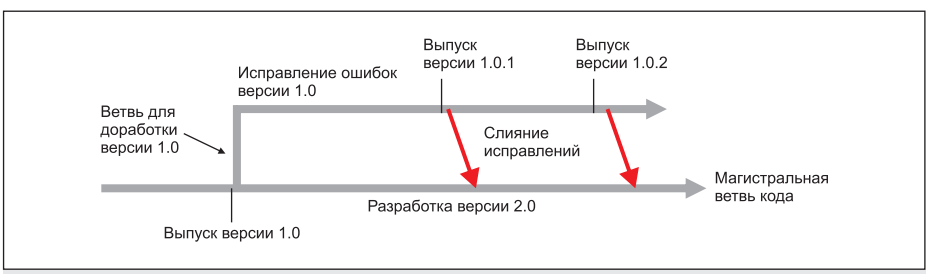

Рис. 12. Повторное слияние кода для исправления дополнительных ошибок в нескольких версиях

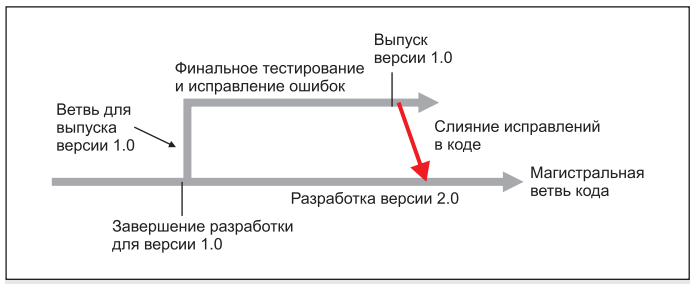

Рис. 13. Слияние кода для исправления ошибок в нескольких версиях, без ожидания результатов тестирования

Такой подход позволяет разделить полномочия и распараллелить потоки исполнения задач разработки. Большая часть команды сможет быстро переключиться на работу над следующей основной выпускаемой версией 2.0, в то время как работы, связанные с тестированием и выпуском предыдущей исправленной версии 1.0.1, останется выполнять небольшая группа специалистов.

Как правило, речь в описанном сценарии идет о незначительном количестве исправлений, обнаруживаемых во время заключительного тестирования в отдельной ветви до выпуска этой версии. Затем, уже после выпуска версии 1.0, любые окончательные изменения кода, внесенные в ходе заключительного тестирования, сливаются обратно в основную ветвь разработки, чтобы он также пришел к современному виду.

Ведение функциональных ветвей

В отличие от всех рассмотренных ранее вариантов ведения версионных ветвей, работа с функциональными ветвями (feature branches), предназначенными для внедрения новых функциональных возможностей, проводится по совсем иному сценарию.

Такие ветви создаются, когда необходимо провести некоторые достаточно сложные работы параллельно со всей остальной разработкой. Например, это может потребоваться при добавлении в приложение новой функциональной возможности, которая достаточно длительное время будет дестабилизировать код магистральной ветви, создавая тем самым проблемы для других разработчиков, не задействованных в реализации этих конкретных возможностей.

Когда разработчики получают в свое распоряжение отдельную ветвь, они могут долго заниматься реализацией новой сложной функциональной возможности. При этом вся их работа будет изолированной и гарантированно не помешает другим сотрудникам, и наоборот (рис. 14).

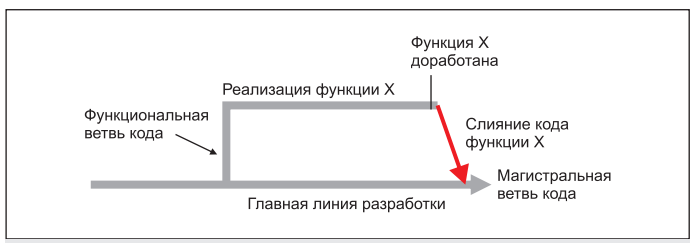

Рис. 14. Ветвление в системе Subversion для внедрения новых функциональных возможностей

Большинство компаний предпочитает, чтобы код, помещаемый в магистральную ветвь репозитория, успешно компилировался и сохранял стабильность ежедневно.

Время от времени в практике разработки программных приложений возникает необходимость совершать достаточно крупные изменения, способные на протяжении нескольких дней или даже недель отрицательно влиять на стабильность кода в магистральной ветви и на возможность его компиляции. Для хранения таких изменений в репозитории удобнее создавать специальные параллельные ветви.

Таким образом, вне зависимости от продвижения разработки столь крупных изменений, код в магистральной ветви всегда остается стабильным и может подвергаться ежедневной компиляции практически в любой момент.

В то же время код таких отдельных ветвей должен достаточно часто синхронизироваться с кодом магистральной ветви, чтобы гарантировать сохранение актуальности и соответствия с другими изменениями в коде, выполняемыми другими подгруппами разработчиков (рис. 15).

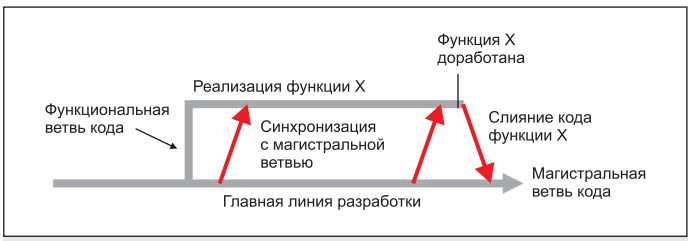

Рис. 15. Регулярная синхронизация кода отдельных ветвей с магистралью для сохранения актуальности изменений

После того как разработка в функциональной ветви завершается, ее содержимое сливается обратно в магистраль. Ветвь, необходимость в которой пропадает, удаляется. В результате в магистральной ветви будут храниться все изменения, выполненные в функциональной ветви, а также все собственные изменения.

В крупных программных проектах распространена практика одновременного существования нескольких параллельных ветвей, в каждой из которых отдельные команды работают над реализацией отдельных компонентов приложения (рис. 16).

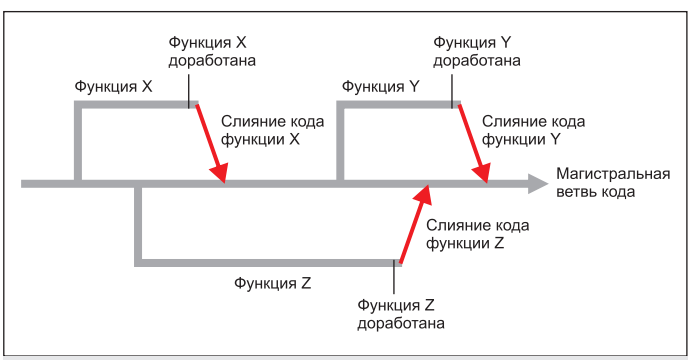

Рис. 16. Одновременное ведение нескольких функциональных ветвей в крупных программных проектах

Подводя итог, можно сказать, что ведение функциональных ветвей является полезным механизмом для минимизации риска вмешательства работы одних команд в действия других.

Как уже упоминалось, рекомендуется достаточно часто объединять последние изменения в коде магистральной ветви с содержимым функциональных ветвей. Это дает уверенность, что изменения кода в функциональных ветвях совместимы и действительно будут работать с последними магистральными изменениями других разработчиков.

Путем разделения процессов разработки крупных компонентов по отдельным параллельным ветвям, менеджмент проекта приобретает возможность систематического и упорядоченного выполнения сложных изменений в приложении. При этом можно не беспокоиться о нежелательном воздействии нескольких разных подразделений разработчиков на работу друг друга.

Поддержка тегов

Одним из важнейших понятий в системах контроля версий заслуженно являются теги. Здесь можно провести аналогию с использованием тегов в службах разрешения имен в сетях, DNS. Идея заключается в том, чтобы обеспечить специалистам удобную возможность использовать знакомые термины при запросе сетевых местоположений. Например, гораздо легче запомнить **www.google.com**, чем запрашивать доступ по его IP-адресу 74.125.91.103.

Аналогично тег в системе SVN представляет собой символическое имя, которое соответствует конкретному номеру ревизии исходного кода в репозитории. Другими словами, тег — это именованный снимок состояния проекта в конкретный момент времени.

Например, можно создавать теги для каждой из завершенных версий продукта, для каждой сборки из числа поставляемых заказчикам или же для каждого состояния кода к моменту проведения производственных совещаний и т. д.

Основная цель применения тегов — идентифицировать состояние репозитория в определенный момент времени, присвоив ему символическое имя для последующих ссылок к коду по данному имени (рис. 17). За счет этого теги значительно упрощают получение нужного состояния репозитория, поскольку разработчики в любой момент могут обращаться к нему по имени тега вместо числового номера ревизии, неудобного для запоминания, ассоциации и ручного набора с клавиатуры.

Так, на момент выпуска бета-версии какого-либо продукта для состояния кода присвоен номер ревизии 2589. Здесь удобно создать какой-либо понятный именованный тег, например v2.1 BETA. Подобный вариант гораздо легче запомнить, чем абстрактный числовой код, а обозначают они одно и то же.

Среди разработчиков считается хорошей практикой создавать теги с описательным именем для каждого важного состояния репозитория, скажем, для каждой альфа- и бета-версии, каждого релиз-кандидата и финального состояния каждой выпускаемой версии продукта.

В качестве дополнения можно упомянуть, что в некоторых системах контроля версий вместо термина «тег» (tag) применяется термин «метка» (label), но по своей сути они означают одно и то же.

Подстановка ключевых слов

Система Subversion способна поддерживать целый ряд полезных ключевых слов, используемых в качестве закладок.

Для повышения удобства работы такие ключевые слова содержатся в файлах проекта, распознаются системой Subversion и автоматически заменяются на динамически генерируемые данные, актуальные в текущем контексте.

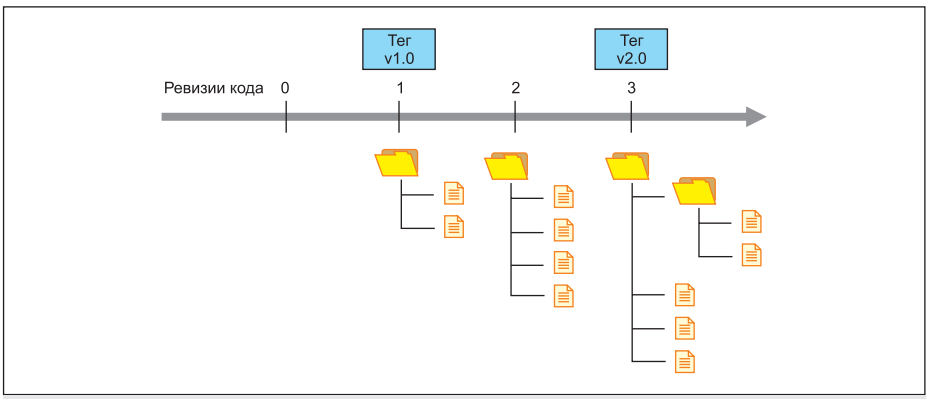

Рис. 17. Использование именованных тегов в Subversion

Предположим, что заголовочный C-файл содержит следующий комментарий:

```
/*
Revision:
$Rev$:

Author:
$Author$:

Date:
$Date$:
*/
```

Здесь вместо конкретных значений номера ревизии (Revision), автора модификации (Author) и даты изменений (Date) указаны некоторые ключевые слова.

После того как файл помещен в репозиторий, эти ключевые слова будут заменены динамически генерируемыми данными, актуальными в конкретной ситуации:

```
/*
Revision:
$Rev: 67 $:

Author:
$Author: mark $:

Date:
$Date: 2011-01-15 08:42:00 -0400 (Sat, 15 Jan 2011) $:
*/
```

Благодаря такой замене при последующем открытии файлов в редакторе исходного кода заголовочный файл будет содержать динамически обновляемую информацию о текущей ревизии данного файла (в приведенном примере — 67), о разработчике, который произвел последнюю модификацию (mark), о дате и времени изменений и т. д. При этом полностью отпадает необходимость заполнять все поля вручную каждый раз перед переносом в репозиторий.

Продолжение следует

Литература

1. Сергеева А. Тестирование работоспособности промышленного компьютера // Компоненты и технологии. 2014. № 2.
2. Сергеева А. Применение реинжиниринга при проектировании встраиваемых систем. Часть 1 // Компоненты и технологии. 2014. № 9.
3. Сергеева А. Применение реинжиниринга при проектировании встраиваемых систем. Часть 2 // Компоненты и технологии. 2014. № 10.
4. Сергеева А. Инструменты тестировщика, или С чего начать новичку // Системный администратор. 2014. № 7–8.
5. Об инструменте Atollic TrueANALYZER. **www.atollic.com/trueanalyzer**
6. Официальный сайт разработчиков Subversion. **www.subversion.apache.org**
7. Сергеева А. Гибкие методологии разработки современных программных приложений // Системный администратор. 2015. № 1–2.
8. Официальное описание работы с Subversion. **www.svnbook.red-bean.com**