Глава 10

Тестирование и управление изменениями

Тестирование и управление изменениями — это не отдельные этапы ЖЦ разработки ПО — они охватывают весь жизненный цикл. Тестирование — это совсем не отладка программ. Артефакты разработки каждого этапа ЖЦ должны быть протестированы. Аналогично, понятие управления изменениями применимо не только в отношении усовершенствований, требуемых заказчиками, или ошибок, обнаруженных в ходе тестирования. Управление изменениями представляет собой фундаментальный аспект управления проектом в целом — запросы на изменения должны быть документированы, а влияние каждого изменения на артефакты разработки должно отслеживаться и перепроверяться после их реализации. В основе тестирования и управления изменениями лежит понятие прослеживаемости. Прослеживаемость обеспечивает фиксацию, связь и отслеживание всех важных артефактов разработки, включая требования. Конечная цель прослеживаемости — дать возможность выработать полную документацию на систему, которая является заведомо корректной и непротиворечивой применительно к различным документам и моделям, начиная с требований и заканчивая технической и пользовательской документацией. Элементами прослеживаемости могут быть текстовые и графические модели. Прослеживаемость устанавливает явные связи между элементами прослеживаемости. Связи могут быть прямыми и косвенными. Они позволяют проводить анализ последствий при изменении любого элемента, находящегося на пути прослеживаемости. Ранее в этой книге мы провели различие между системными услугами и системными ограничениями.

Прослеживаемость, тестирование и управление изменениями часто ассоциируются с системными услугами, которые заявляют о себе в требованиях, выражаемых прецедентами. Однако, не следует забывать о необходимости отслеживания того, как выдерживаются системные ограничения, и, соответственно, тестирования и управления их изменениями.

10.1. Тестирование системных сервисов

Скэч (Schach) [77] проводит различие между неформальным и методическим тестированием системных сервисов. Каждый разработчик выполняет *неформальное тестирование* при моделировании или реализации системных сервисов. По своему харак-

теру неформальное тестирование несовершенно. Тот, кто разрабатывает сервис, менее всего заинтересован в том, чтобы в программе, реализующей этот сервис, были обнаружены ошибки.

Значение неформального тестирования незначительно, и оно должно быть дополнено *методическим тестированием*. Существует два основных вида методического тестирования [77].

1. Тестирование без выполнения программы (формальные пересмотры) (formal reviews)
   - Сквозной контроль
   - Инспекция
2. Тестирование, основанное на выполнении программы
   - Тестирование по спецификации
   - Тестирование по программному коду

10.1.1. Сквозной контроль

Сквозной контроль (*walkthrough*) представляет собой один из видов формального пересмотра артефактов методом "мозгового штурма", который может проводиться на любом этапе разработки. Это дружественная встреча разработчиков, тщательно спланированная, с ясно определенными целями, повесткой дня, продолжительностью и составом участников. Многие бригады ИТ-разработчиков проводят сквозной контроль еженедельно. *За несколько дней до совещания по сквозному контролю* участникам раздаются материалы (описания моделей, документы, программы и т.д.), которые подлежат пересмотру на совещании. Материалы участникам собирает и раздает модератор сквозного контроля. Участники изучают материалы и передают модератору свои замечания еще до начала совещания. Само совещание относительно непродолжительно (самое большее два-три часа). *Во время совещания* модератор представляет замечания и открывает обсуждение по каждому вопросу. Цель освещения состоит в том, чтобы уточнить проблему, а не выводить разработчиков "на чистую воду"! Разработчик, который стоит за проблемой, в данном случае не важен и может вообще оставаться неизвестным (хотя обычно это не так). Общая идея состоит в том, чтобы подтвердить существование проблемы. При этом не следует предпринимать попыток решить проблему. О полезности и эффективности сквозного контроля свидетельствуют многие факты. Он придает процессу разработки четкость и профессионализм, вносит существенный вклад в достижение высокой продуктивности и способствует выполнению работы в срок, имеет очень важное значение с точки зрения информированности участников проекта и, в конечном итоге, позволяет повысить качество разрабатываемого ПО.

10.1.2. Инспекция

Подобно сквозному контролю *инспекция* (*inspection*) представляет собой совещание, проводимое в дружелюбном духе, но под пристальным наблюдением со стороны руководства проекта. Ее целью также является выявление дефектов, подтверждение того, что они действительно являются дефектами, их фиксация и назначение сроков их устранения и лиц, ответственных за это.

В отличие от сквозного контроля инспекции проводятся не так часто, могут быть посвящены только отдельным, имеющим критическое значение, вопросам и носят более формальный и более строгий характер. Инспекция организована в виде нескольких этапов. Она начинается с *этапа планирования*, на котором определяются участники инспекции и целевая область инспектирования.

До начала инспекционного заседания необходимо провести короткое *информационное совещание*. Во время информационного совещания разработчик, продукт которого подлежит инспектированию, вводит участников в курс дела. Инспекционные материалы раздаются участникам во время или перед информационным совещанием.

Информационное совещание обычно проводится за неделю до инспекционного совещания. Это дает инспекционной бригаде время изучить материалы и подготовиться к совещанию. Во время совещания дефекты идентифицируются, фиксируются и нумеруются. Сразу после совещания модератор готовит *журнал дефектов* — в идеале он ведется с использованием *средств управления изменениями*, связанными с проектом.

Обычно от разработчика требуется быстро решить проблемы, связанные с дефектами, и зафиксировать принятое решение с помощью средств управления изменениями. Модератор должен проверить отчет об устранении дефекта и решить, требуется ли *повторное инспектирование*. В случае если он удовлетворен решением проблемы, модератор, проконсультировавшись с руководителем проекта, направляет разрабатываемый модуль *группе обеспечения качества ПО* (*Software Quality Assurance — SQA*) организации (если такая группа существует).

Группа SQA должна состоять из лучших специалистов, имеющихся в организации. Группа никак не должна быть связана с проектом за исключением своей роли по обеспечению качества. Группа (а не истинные разработчики!) несет ответственность за конечное качество продукта.

10.1.3. Тестирование по отношению к спецификации

Тестирование по отношению к спецификации представляет собой одну из разновидностей тестов, основанных на выполнении (прогоне) программы. Оно применимо к исполняемым программным продуктам, а не документам или моделям. Этот вид тестирования также известен под другими названиями, такими как *тестирование по методу черного ящика*, функциональное тестирование, тестирование по входу-выходу и т.д.

Принцип тестирования по отношению к спецификации заключается в том, что разработчик рассматривает тестируемый модуль как "черный ящик", на вход которого подается некоторая информация и который вырабатывает некоторый выход. При этом не делается никаких попыток понять программную логику или вычислительные алгоритмы.

При тестировании по отношению к спецификации необходимо выработать *тестовые требования*, исходя из *требований-прецедентов*, а затем идентифицировать и зафиксировать их в отдельных документах, представляющих тест-план и тестовые прецеденты. Эти документы предоставляют в распоряжение специалиста по тестированию *сценарий тестирования* (*test scenario*). Сценарий можно записать с помощью специальных средств автоматизации тестирования, а затем использовать его многократно. Подобные средства получили название средств "записи-воспроизведения". Такие средства особенно полезны в случае регрессионного тестирования (разд. 1.3.10).

Тестирование по отношению к спецификации подходит для выявления дефектов, которые трудно "поймать" другим способом. В частности, тестирование по спецификации позволяет обнаружить *пропущенные функции* — что-то, что (надо надеяться) было

документировано как требование-прецедент (а следовательно, и тестовое требование), однако, так и не было запрограммировано. Этот вид тестирования позволяет также выявить пропущенные функции, которые не были оформлены документально в виде прецедентов, но явно пропущены в реализации системы.

10.1.4. Тестирование по отношению к программному коду

Тестирование по отношению к программному коду представляет собой второй вид тестирования, основанный на выполнении программы. Он также известен под названиями *тестирования по методу прозрачного ящика*, тестирование по методу стеклянного ящика, тестирование путем покрытия логики программы и тестирование путем покрытия путей программы. Тестирование по отношению к программному коду начинается с тщательного анализа алгоритмов программы. С целью *испытания программы* выводятся прецеденты — это дает определенные гарантии проверки всех возможных выполняемых ветвей программы. Для выполнения программы специальным образом продумывается организация данных.

Для поддержки тестирования по отношению к программному коду также существуют средства "записи-воспроизведения" сценариев тестирования, используемые для регрессионного тестирования. Однако, характер тестирования по коду требует привлечения к использованию этих средств программистов. Многие сценарии "проигрывания" тестов должны быть написаны программистами, а не сгенерированы средствами автоматизации. Даже если сценарии и генерируются автоматически, может потребоваться существенно модифицировать их с помощью программистов.

Подобно всем прочим видам тестирования, основанным на выполнении программы, тестирование по отношению к программному коду не может носить исчерпывающего характера из-за комбинаторного взрывоподобного роста количества возможных тестовых прецедентов даже при умеренном росте сложности программы. Даже если существует возможность тестирования каждой выполняемой ветви программы, нет никаких гарантий, что все дефекты будут обнаружены. Старая мудрость тестирования по-прежнему остается в силе — тестирование способно устранить некоторые ошибки, но не в состоянии доказать правильность программы!

10.2. Тестирование системных ограничений

Тестирование системных ограничений преимущественно факта реализации ограничений в соответствии с перечнем требований и тестовой документацией основано на выполнении программы. Его цель состоит в установлении ограничений. Тестирование системных ограничений включает ряд вопросов, некоторые из них приводятся ниже.

- Тестирование пользовательского интерфейса.
- Тестирование баз данных.
- Тестирование контроля доступа.
- Тестирование производительности.
- Тестирование в утяжеленном режиме.
- Тестирование при отказе.
- Конфигурационное тестирование.
- Инсталляционное тестирование.

Первые два типа тестирования системных ограничений — *тестирование пользовательского интерфейса и баз данных* — очень тесно связаны с тестированием системных услуг. Они обычно проводятся в параллель с тестированием системных ограничений. Поэтому включаются в тестовые документы (разд. 10.3), вырабатываемые для тестирования системных услуг.

10.2.1. Тестирование пользовательского интерфейса

Тестирование GUI-интерфейса пронизывает весь процесс разработки ПО. Оно начинается на ранней стадии этапа выработки требований, сопровождая такие виды деятельности, как работа с архивными документами, включение эскизов окон в документы описания прецедентов и разработка прототипов GUI-интерфейса. Эти ранние тесты GUI-интерфейса концентрируются на удовлетворении функциональных требований и удобстве использования приложения.

Позже, после реализации системы требуется *послереализационное тестирование GUI-интерфейса*. Вначале тестирование проводится разработчиками, затем специалистами по тестированию и — перед выпуском ПО — заказчиками (*пилотное тестирование*) . Ниже приведен примерный перечень вопросов тестового документа, разработанного для послереализационного тестирования [9].

- Соответствует ли название окна его функции?
- Является ли окно модальным или немодальным? Каким оно должно быть?
- Введено ли визуальное различие между обязательными и необязательными полями?
- Можно ли изменить размеры окна, переместить его и восстановить? Необходимо ли это?
- Пропущены ли какие-либо поля?
- Есть ли орфографические ошибки в названиях, метках, именах приглашений для ввода и т.д.?
- Нет ли противоречий в использовании командных кнопок (OK, Cancel, Save, Clear и т.д.) в каких-либо диалоговых окнах?
- Возможно ли всегда отменить текущую операцию (включая операцию удаления)?
- Все ли статические поля защищены от изменения пользователем? Может ли приложение изменить статический текст, корректно ли это делается?
- Применяются ли для статических текстовых полей согласованные шрифты и размеры?
- Соответствуют ли размеры полей редактирования диапазону принимаемых ими значений?
- Все ли поля редактирования инициализированы верными значениями при открытии окна?
- Проверяются ли значения, вводимые в поля редактирования, клиентской программой?
- Правильно ли заполняются из базы данных значения выпадающих списков?

- Используются ли маски редактирования в полях ввода в соответствии с определением?
- Доходчивы ли сообщения об ошибках и легко ли работать с ними?

10.2.2. Тестирование баз данных

Аналогично тестированию GUI-интерфейса тестирование баз данных связано со многими другими видами тестирования. Так, тестирование по методу черного ящика (тестирование по отношению к спецификации) зачастую основано на использовании входной и выходной информации для базы данных. Однако, это не исключает необходимости в отдельной методике тестирования баз данных.

Послереализационное тестирование баз данных включает всестороннее тестирование по методу прозрачного ящика (в соответствии с программным кодом). Наиболее существенной частью тестирования баз данных является *тестирование транзакций*. Тестирование некоторых других аспектов баз данных — производительности, параллелизма, проверка полномочий пользователей — иногда выделяют в отдельные группы тестов.

Аналогично тестированию GUI-интерфейса некоторые тесты баз данных требуется проводить повторно для всех различных функций приложения. Вопросы, требующие рассмотрения при тестировании баз данных, необходимо оформить в виде общего документа. Этот общий документ следует затем присовокупить к описанию всех функциональных тестов (тестов системных услуг). Ниже приводится примерный перечень вопросов, на которые следует обратить внимание при тестировании баз данных [9].

- Проверить, выполняется ли транзакция надлежащим образом при правильных входных данных. Корректна ли обратная связь системы с GUI-интерфейсом? Корректно ли содержимое базы данных после транзакции?
- Проверить, выполняется ли транзакция надлежащим образом при неправильных входных данных. Корректна ли обратная связь системы с GUI-интерфейсом? Корректно ли содержимое базы данных после транзакции?
- Прервать транзакцию до ее завершения. Корректна ли обратная связь системы с GUI-интерфейсом? Корректно ли содержимое базы данных после транзакции?
- Запустить одну и ту же транзакцию параллельно во многих процессах. Сознательно заставить одну из транзакций заблокировать ресурс, необходимый другим транзакциям. Получают ли пользователи понятное объяснение ситуации? Корректно ли содержимое базы данных после транзакций?
- Извлечь каждый из клиентских SQL-операторов из клиентской программы и выполнить их над базой данных в интерактивном режиме. Совпадают ли полученные результаты с ожидаемыми, а также соответствуют ли они результатам при выполнении SQL-операторов в составе программы?
- Выполнить интерактивное тестирование каждого более сложного SQL-запроса (выдаваемого из клиентской программы или хранимой процедуры) по методу прозрачного ящика, включая внешние соединения, объединения, подзапросы, значения типа null, функции агрегации и т.д.

10.2.3. Тестирование авторизации

Тестирование авторизации можно рассматривать как естественное расширение тестирования первых двух типов системных ограничений. Как клиентские (пользовательский интерфейс), так и серверные (базы данных) объекты должны быть защищены от несанкционированного использования. Тестирование авторизации должно обеспечить проверку того, что механизмы безопасности, встроенные в клиентскую и серверную части системы, в действительности защищают систему от несанкционированного проникновения.

Хотя нарушение безопасности безусловно сказывается на базах данных, защита начинается с клиентской части системы. Пользовательский интерфейс программы должен быть в состоянии динамически настраивать собственную конфигурацию в соответствии с уровнем полномочий текущего пользователя (который подтверждается (*аутентифицируется*) идентификатором и паролем пользователя). Пункты меню, командные кнопки и даже целые окна могут стать недоступны пользователям, если у них нет соответствующих прав.

Не все "проходы" в защите системы могут быть обращены в сторону пользователей. Поддержка авторизации является существенной компонентой любой СУБД. Пользователю могут быть предоставлены выборочные права доступа к серверным объектам, при этом *права* (*полномочия*) пользователя по отношению к *серверу* распадаются на две категории.

- Доступ к отдельным *серверным объектам* (таблицам, представлениям, столбцам, хранимым процедурам и т.д.).
- Выполнение SQL-операторов (`select`, `update`, `insert`, `delete` и т.д.).

Полномочия для пользователей можно назначать непосредственно на уровне пользователей или на групповом уровне. Группы позволяют администратору системы защиты назначать права доступа группе пользователей с помощью однократного ввода соответствующих параметров. Пользователь может как не принадлежать ни к одной группе, так и принадлежать к нескольким группам.

Для обеспечения большей гибкости при работе с авторизацией большинство СУБД вводят дополнительный уровень авторизации — *ролевой уровень*. Роли позволяют администратору системы защиты назначать права доступа всем пользователям, которые играют определенную роль в организации. Роли могут носить вложенный характер, т.е. права, предоставленные разным ролевым именам, могут перекрываться.

Для больших приложений ИС *проектирование авторизации* требует скрупулезной работы. Зачастую наряду с прикладной базой данных создается база данных авторизации для хранения и манипулирования клиентскими и серверными полномочиями. После входа пользователя в систему прикладная программа обращается к базе данных, чтобы установить уровень полномочий пользователя и настроить свою конфигурацию применительно к этому пользователю.

Любые изменения в правах доступа к базам данных осуществляются через базу данных авторизации — т.е. никто, включая администратора системы защиты, не имеет возможности непосредственно изменить права доступа к базе данных без предварительного обновления базы данных авторизации. На рис. 10.1 приведен пример проекта базы данных авторизации.

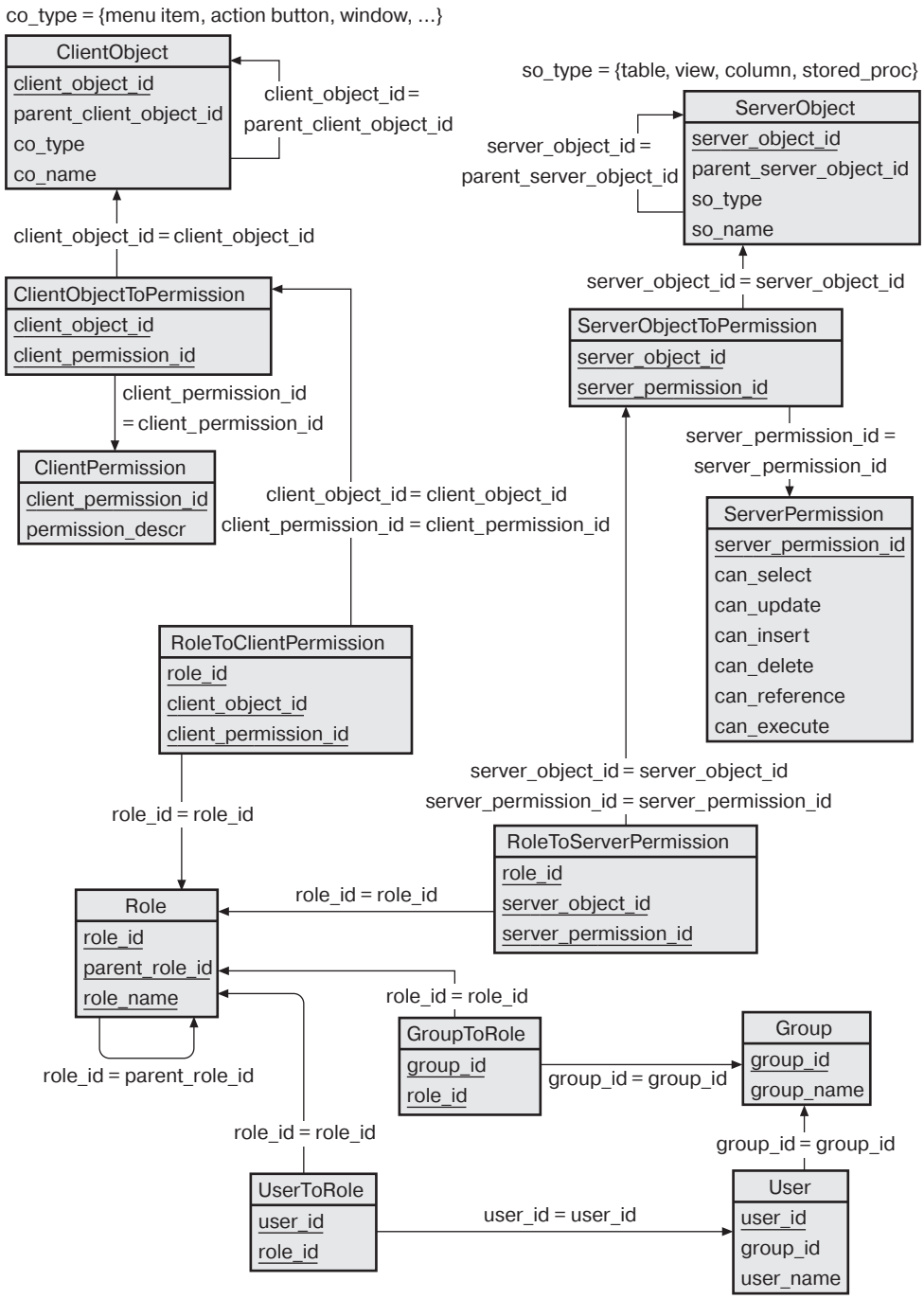

Рис. 10.1. Проект базы данных авторизации

10.2.4. Тестирование других ограничений

Помимо описанных выше тестирование системных ограничений включает следующие виды тестирования.

- Тестирование производительности.
- Тестирование в утяжеленном режиме.
- Тестирование при отказе.
- Конфигурационное тестирование.
- Инсталляционное тестирование.

Тестирование производительности направлено на измерение ограничений производительности, требуемых заказчиком. Ограничения связаны со *скоростью транзакций* и *пропускной способностью*. Тестирование проводится при различной рабочей загрузке систем, включая предполагаемую *пиковую загрузку*. Тестирование производительности является важной составляющей *настройки системы*.

Тестирование в утяжеленном режиме проектируется таким образом, чтобы вывести из стоя систему при предъявлении к ней завышенных требований — из-за недостатка ресурсов, необычной конкуренции за ресурсы, непредусмотренной частоты, величины или объема требований к ресурсам. Тестирование в утяжеленном режиме часто сочетается с тестированием производительности и может потребовать соответствующей аппаратной и программной *оснастки*.

Тестирование при отказе направлено на изучение реакции системы на различные аппаратные, сетевые или программные сбои. Этот вид тестирования тесно связан с процедурами восстановления, поддерживаемыми СУБД.

Конфигурационное тестирование связано с проверкой функционирования системы при различной аппаратной и программной конфигурации. Для большинства производственных сред предполагается, что система способна функционировать на различных клиентских рабочих станциях, которые подключаются к базе данных с использованием различных сетевых протоколов. На клиентских рабочих станциях может быть инсталлировано различное ПО (например, драйверы), которое может конфликтовать с предусмотренными установками.

Инсталляционное тестирование является расширением конфигурационного тестирования. Оно связано с проверкой надлежащего функционирования системы на каждой из платформ, на которых она инсталлируется. Это означает фактическое повторение тестирования системных услуг.

10.3. Документация по тестированию и управлению изменениями

Документация по тестированию и управлению изменениями составляет неотъемлемую часть остальной системной документации, включая документацию по прецедентам, рис. 10.2. Системные *функции*, определенные в модели бизнес-прецедентов (разд. 3.5.2), можно использовать для написания первоначального тест-плана. Затем модель прецедентов используется для написания документации по тест-прецедентам и определения *тестовых требований*. Дефекты, обнаруженные во время тестирования, фиксируются в *документации по дефектам*. В документации по усовершенствованию отражаются все не-

реализованные *требования-прецеденты*. При использовании CASE-средств у разработчиков существуют следующие возможности по созданию документации.

- Разработка описательных документов и их последующее использование для создания требований (тестовые требования, требования-прецеденты и т.д.) в CASE-репозитории.
- Использование CASE-средств для ввода требований в репозиторий и последующая генерация документации на их основе.

На рис. 10.3 показан фрагмент документации по тест-прецедентам, используемой для ввода тестовых требований в репозиторий. Аналогично требованиям-прецедентам тестовые требования нумеруются и организуются в виде иерархии. Некоторые тестовые требования непосредственно соответствуют требованиям-прецедентам. Отсюда основной раздел документа называется "Соответствие спецификации прецедентов".

Остальные разделы документа по тестовым прецедентам должны идентифицировать требования к тестированию GUI-интерфейса, тестированию баз данных и тестированию повторно используемых компонент. Этот вид тестов следует включить в виде составной части в тестовые или функциональные модули по двум причинам. Во-первых, для тестирования GUI-интерфейса, баз данных и общих компонент (например, таких, которые оформляются в виде библиотек DLL) нам необходим функциональный контекст, в рамках которого имеют смысл входные и выходные данные. Во-вторых, дефекты GUI, баз данных и общих компонент могут проявиться только в контексте некоторых, но далеко не всех функциональных тестов.

Документация по тестовым прецедентам используется для фиксации результатов. Отсюда три столбца после каждого тестового требования, показанные на рис. 10.3. Тестовое требование может завершиться неудачным тестированием, пройти испытание условно или безусловно

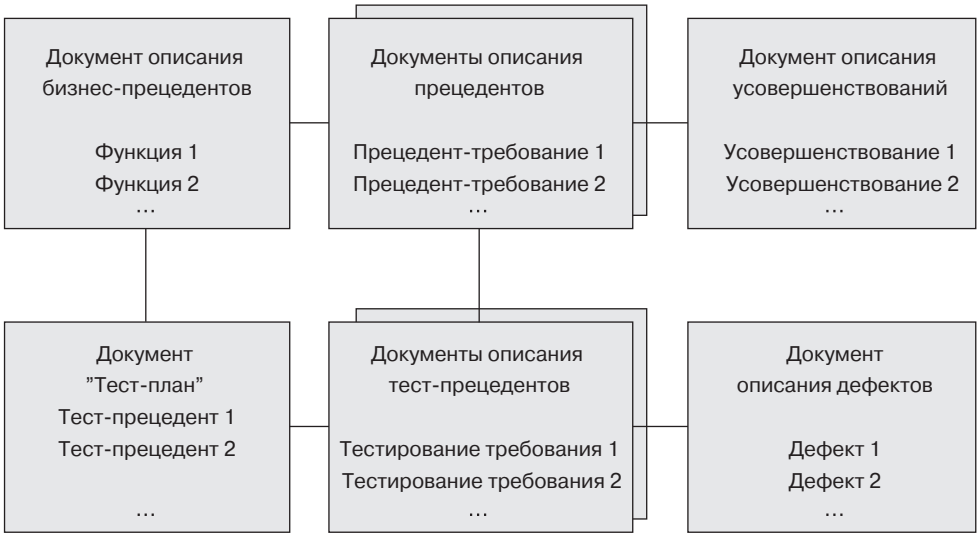

Рис. 10.2. Документация по тестированию и управлению изменениями

Рис. 10.3. Документация по тест-прецедентам

10.4. Управление изменениями

Тестирование приводит к обнаружению *дефектов*. Дефекты необходимо исправлять. Чтобы исправить дефекты, их необходимо отправить в виде *запросов на изменение* (*change requests*) и адресовать разработчикам. Некоторые запросы на изменение связаны с *усовершенствованием*, а не исправлением дефектов. Как дефекты, так и усовершенствования изменяют свой статус, им могут быть присвоены различные приоритеты, за их сопровождение могут отвечать различные лица, их необходимо отслеживать в связи с их источниками в виде документации по тестированию и прецедентам и т.д.

Управление изменениями для любого программного проекта, в котором принимает участие большое количество разработчиков, представляет собой сложную и ответственную задачу. Рассмотрим сценарий, в рамках которого исправление двух различных дефектов поручено двум различным разработчикам, при этом оказалось, что исправление этих на первый взгляд несвязанных дефектов требует внесения изменений в одну и ту же программную компоненту. До тех пор, пока разработчики не будут знать об этом возможном конфликте, оба разработчика могут одновременно работать

над исправлением, и в конце концов более позднее исправление приведет к отмене исправления, сделанного раньше.

Для надлежащего управления изменениями необходимо применение *средств управления запросами на изменения* (как правило, входящих в состав CASE-средств). Подобные средства обеспечивают возможность интерактивного управления изменениями и гарантируют работу всех разработчиков с последними версиями документов. Изменения, внесенные в документ одним из участников проекта, тут же становятся достоянием остальных разработчиков. Потенциальные конфликты разрешаются с помощью механизмов блокировки или управления версиями. В первом случае заблокированный документ становится временно недоступен другим разработчикам. В последнем случае возможно создание нескольких версий одного документа, а конфликты между версиями разрешаются позднее посредством согласования.

10.4.1. Отправка запроса на изменение

Обычно *запрос на изменение* связан либо с дефектом, либо с усовершенствованием. Запрос на изменение вводится в проектный репозиторий. После ввода в репозиторий разработчики могут отслеживать продвижение запроса на изменение, наблюдать за его статусом и действовать в соответствии с ним. *Действия*, выполняемые над запросом на изменение, зависят от текущего статуса запроса.

На рис. 10.4 показана основная вкладка диалогового окна ввода информации о дефектах [69]. Дефекты нумеруются и подробно описываются.

Рис. 10.4. Окно подготовки информации для отправки запроса на изменение

Информацию о приоритете, серьезности, проекте и ответственном исполнителе можно ввести с помощью выбора подходящих значений выпадающего списка

(значения атрибутов, содержащихся в выпадающем списке и в других местах формы, можно настроить в соответствии с нуждами проекта). Остальные поля позволяют вводить описательную информацию, включая возможность присоединения документации, имеющей отношение к дефекту, в частности, фрагментов программного кода.

Действие по *отправке* запроса на изменение может привести к автоматическому уведомлению участников проектной бригады через электронную почту. После этого запрос на изменение переходит в *состояние Submitted*. Руководство проекта может настроить инструментальные средства таким образом, чтобы в каждом состоянии выполнялись заранее определенные действия. Например, в состоянии `Submitted` могут быть допустимы следующие действия: `Assign` (Назначить [задание]) (участнику бригады), `Modify` (Модифицировать (некоторые детали запроса)), `Postpone` (Отложить), `Delete` (Удалить) (без исправления), `Close` (Закрыть) (возможно, в результате исправления).

10.4.2. Отслеживание запросов на изменение

Каждый запрос на изменение назначается члену бригады. Член бригады может открыть (`Open`) запрос на изменение.

Рис. 10.5. Диаграмма измерений

Если запрос находится в состоянии *Open*, никто из других членов бригады не может модифицировать статус запроса. После разрешения проблемы, связанной с запросом на изменение, разработчик может выполнить над ним действие `Resolve`. Подробности решения можно ввести в форму и отправить по электронной почте уведомление руководству проекта и специалистам по тестированию. Последним может потребоваться выполнить действие по верификации (`Verify`) разрешенного запроса на изменение.

На любом этапе средства управления изменениями могут отследить запросы и выработать легкие для понимания диаграммы и отчеты (*проектные показатели* (*project metrics*)). Диаграммы и отчеты могут содержать данные по оценке количества не распределенных по членам бригады дефектам помогают выявить загрузку каждого члена бригады, показать, сколько осталось неразрешенных дефектов и т.д.

На рис. 10.5 показана диаграмма распределения актуальных дефектов по приоритетам. Шесть дефектов должны быть разрешены немедленно, на пятьдесят пять следует обратить более пристальное внимание, шестьдесят семь находятся в обычной очереди и шестьдесят восемь имеют низкий приоритет.

10.5. Прослеживаемость

Прослеживаемость, тестирование и управление изменениями — не самоцель, и их значение не следует переоценивать. Разработчики должны концентрироваться на разработке, а не на отслеживании, тестировании или управлении изменениями. С этими проблемами, как правило, связаны значительные проектные расходы. Однако, в долгосрочной перспективе *отсутствие управления* этими проблемами влечет не менее значительные затраты.

Поскольку основу тестирования и управления изменениями составляет прослеживаемость, рамки и глубину прослеживаемости проекта следует определять с использованием *анализа затрат и результатов*. По меньшей мере прослеживаемость следует поддерживать между требованиями на основе прецедентов и дефектами. В более развитой модели между требованиями-прецедентами и дефектами в маршрут прослеживаемости можно добавить тестовые требования. В еще более сложной модели прослеживаемости рамки прослеживаемости могут включать системные функции, тестовые прецеденты, усовершенствования, точки тестовой верификации и другие артефакты разработки ПО.

Оставшаяся часть этой главы посвящена рассмотрению модели прослеживаемости, которая соответствует связям между системной документацией, показанной на рис. 10.2. В документации на бизнес-прецеденты перечислены *системные функции*. Документ, описывающий тест-план, идентифицирует *тестовые прецеденты*. *Функции* связаны с *тестовыми прецедентами* и *требованиями-прецедентами*, что отражено в документации по прецедентам. *Тестовые требования*, представленные в документе по тестовым прецедентам, можно проследить назад к *тестовым прецедентам* и *требованиям-прецедентам*. *Тестовые требования*, связанные с *дефектами* и усовершенствованиями, прослеживаются до *прецедентов-требований*. Прослеживать *дефекты* до *усовершенствований* не требуется.

10.5.1. Прослеживаемость от системных возможностей к прецедентам и прецедентным требованиям

Функциональные возможности системы представляют собой общую часть функций, которые должны быть реализованы в системе. Это бизнес-процесс, представляющий собой существенную часть системы с точки зрения ее эффективности. Обычно сис-

темные функциональные возможности соответствуют *бизнес-прецедентам* модели бизнес-прецедентов (разд. 3.5.2). Если модель бизнес-прецедентов формально не разрабатывалась, системные возможности идентифицируются в документе, отражающем стратегическое видение системы (в технологии Rational Rose подобный документ так и называется — *Vision. Прим. ред.*)

Каждая системная возможность реализуется с помощью *требований-прецедентов*, выраженных в виде одного или нескольких *прецедентов*. Прослеживание прецедентов обратно к потребностям заказчиков (выраженные в системных возможностях) помогает проверить обоснованность и правильность модели прецедентов. Эта стратегия "очерчивает рамки" фиксации требований и способствует выполнению этапа установления требований. Она также помогает осуществлению наращиваемой разработки и предоставлению программного продукта заказчикам.

Данная стратегия может стать источником проблем, если требования-прецеденты в рамах каждого прецедента связаны с функциями только косвенно. Это может привести к ситуации, при которой между функцией и прецедентом существует траектория, хотя большая часть требований-прецедентов не имеет отношения к функции. Принятие решения о том, обоснованно ли существование траектории между функцией и прецедентом, может оказаться тяжелой и неблагодарной задачей.

Рис. 10.6. Прослеживаемость от функций до прецедентов и требований-прецедентов

Чтобы избежать масштабирования и долговременных проблем, связанных с этой стратегией, матрица прослеживаемости должна отслеживать функции не только по

отношению к прецедентам, но также непосредственно по отношению к требованиям-прецедентам. Это возможно в том случае, если сам прецедент рассматривается как обобщенное требование-прецедент, под которым находится иерархия специализированных требований-прецедентов.

Это показано на рис. 10.6. Столбцы содержат прецеденты и требования-прецеденты в рамках прецедентов. Иерархическое представление требований-прецедентов можно свернуть или раскрыть. Стрелки обозначают траектории от функций к прецедентам и требованиям-прецедентам. Некоторые стрелки перечеркнуты. Это *подозрительные траектории* (*suspect traces*). Траектория становится подозрительной, когда она направлена от или к изменению требований. Прежде, чем разорвать подозрительные связи, разработчику следует проанализировать их.

10.5.2. Прослеживаемость от тест-плана к тест-прецедентам и тестовым требованиям

Документ описания *тест-плана* играет для тест-прецедентов ту же роль, что документ описания бизнес-прецедентов для прецедентов. Тест-план идентифицирует обобщенную проектную информацию и программные компоненты (*тест-прецеденты*), которые требуется протестировать. Тест-план описывает также стратегию тестирования для проекта, необходимые для тестирования ресурсы, усилия и затраты.

Рис. 10.7. Прослеживаемость от тест-плана к тест-прецедентам и тестовым требованиям

Каждый тест-прецедент, идентифицированный в тест-плане, должен быть описан в виде документа по тест-прецеденту. Отображение тестовых требований на тест-прецеденты и тест-планы дает преимущества, аналогичные преимуществам применения прослеживаемости между функциями, прецедентами и требованиями-прецедентами — ограничение области тестирования, масштабируемость и т.д.

На рис. 10.7 показана матрица прослеживаемости от тест-плана к тест-прецедентам и тестовым требованиям в рамках тест-прецедентов. Иерархическое представление требований-прецедентов можно свернуть или раскрыть.

10.5.3. Прослеживаемость от UML-диаграмм к документам и требованиям

Прослеживаемость и управление требованиями применимо не только к описательным документам и требованиям, представленным в текстовом виде, которые хранятся в CASE-репозитории. Репозиторий хранит также UML-модели. Графические объекты UML-диаграмм могут быть связаны гиперссылками с документами и требованиями.

Прослеживаемость от визуальных артефактов UML к любым другим записям репозитория (в частности, документам и требованиям) можно установить для различных *графических пиктограмм* UML. Пожалуй, наиболее важными из этих пиктограмм являются пиктограммы прецедентов на диаграммах прецедентов.

Рис. 10.8. Установление в документе гиперссылок на графические пиктограммы прецедентов

На рис. 10.8 показано диалоговое окно для установления гиперссылок между графическими пиктограммами прецедента (`Maintain Ads`) и документом. Гиперссылка устанавливается изнутри UML-диаграммы прецедентов. В качестве связанного документа может выступать любой документ репозитория, включая документы, показанные на рис. 10.2.

На рис. 10.9 показано диалоговое окно для установления гиперссылок между графическими пиктограммами прецедента (`Maintain Ads again`) и требованием-прецедентом. В общем случае возможно установить связь между пиктограммой и любым типом требования.

Рис. 10.9. Установление гиперссылок требования на графические пиктограммы прецедентов

10.5.4. Прослеживаемость от требований-прецедентов к тестовым требованиям

Прослеживаемость между требованиями-прецедентами и тестовыми требованиями является критическим фактором при оценке того, удовлетворяет ли приложение бизнес-требованиям, установленным для него. Связи между этими двумя типами требований позволяют пользователю прослеживать дефекты через тестовые требования назад к требованиям-прецедентам и системным функциям (см. рис. 10.2).

На рис. 10.10 показана матрица прослеживаемости, включающая траектории между требованиями-прецедентами и тестовыми требованиями. Обратите внимание, что как требования-прецеденты, так и тестовые требования организованы иерархически. Иерархические уровни, на которых определяются траектории, могут быть определены заранее.

Рис. 10.10. Прослеживаемость от требований-прецедентов к тестовым требованиям

10.5.5. Прослеживаемость от требований к дефектам

Документация по тест-прецедентам представляется в виде сценариев, содержащих тестовые требования, которые требуют верификации при тестировании. Сценарии используются при ручном тестировании, но многие из этих сценариев можно автоматизировать для использования в тестовых средствах "записи-воспроизведения". Тестовые требования в документации по тест-прецедентам можно затем использовать для установления точек верификации в этих автоматизированных тестах.

Точка верификации (*verification point*) — это требование, содержащееся в сценарии, которое используется (при *регрессионном тестировании*) для подтверждения состояния тестового объекта в рамках различных версий (скомпонованных версий) тестируемого приложения. Существуют различные типы точек верификации [69]. Точка верификации может быть установлена для проверки отсутствия изменений в тексте, точности числового значения, идентичности двух файлов, отсутствия изменений в пунктах меню, соответствия полученных результатов вычислений ожидаемым и т.д.

Автоматизированное тестирование требует работы с двумя файлами данных — базисным файлом данных и файлом фактических данных. Во время *записи* процедура точки верификации записывает информацию об объекте в *базисный файл данных*. Эта информация представляет собой базис для сравнения при последующих тестах (*воспроизведении*). Резуль-

таты того сравнения запоминаются в *файле фактических данных*. Результаты каждой *неудачной процедуры точки верификации* требуют дальнейшего изучения и, при необходимости, внесения в репозиторий средств управления изменениями в качестве *дефекта*.

Безусловно все дефекты, обнаруженные с помощью средств автоматизации или вручную, должны быть сопоставлены с тестовыми требованиями. На рис. 10.11 показано средство, которое отображает все дефекты с помощью строкового броузера в верхней части окна. Текущий выбранный дефект можно сопоставить с одним или более тестовым требованием. В приведенном примере показана траектория от двух тестовых требований к выделенному дефекту.

10.5.6. Прослеживаемость от требований-прецедентов к усовершенствованиям

Дефекты необходимо прослеживать непосредственно до тестовых требований. Усовершенствования (которые должны быть реализованы в последующих выпусках продукта) должны содержать соответствующие пояснения о требованиях-прецедентах. Изредка, в ситуации, когда дефект был преобразован в усовершенствование, связь по прослеживаемости между требованиями-прецедентами и тестовыми требованиями может позволить пользователю проследить путь дефекта до усовершенствования.

На рис. 10.12 продемонстрировано то же средство (см. рис. 10.11), которое можно использовать для управления усовершенствованиями и дефектами, как, впрочем, и для любых других типов запросов на изменения.

Рис. 10.11. Прослеживаемость от тестовых требований к дефектам

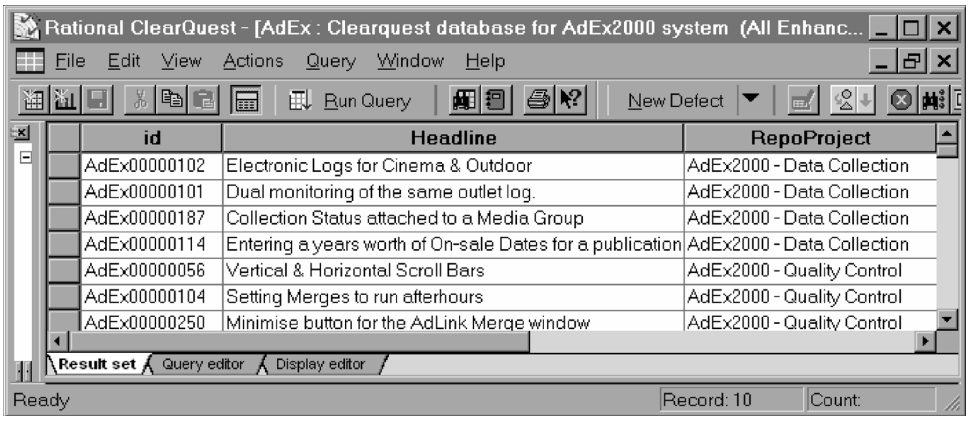

Рис. 10.12. Усовершенствования

Резюме

В этой последней главе книги мы обратились к вопросам тестирования и управления изменениями. Деятельность по тестированию и управлению изменениями охватывает весь ЖЦ разработки ПО, однако, наибольшее внимание ей уделяется ближе к завершению проекта. Тестирование и управление изменениями предполагает существование связей по прослеживаемости между системными артефактами, которые надлежащим образом поддерживаются в течение всей разработки.

Тестирование подразделяется на тестирование системных услуг и тестирование системных ограничений. *Тестирование системных услуг* может быть основано на выполнении программы или проводиться без выполнения программы. *Тестирование без выполнения программы* включает сквозной контроль и инспекции. *Тестирование с прогоном программы* может быть тестированием по отношению к спецификации или тестированием по отношению к программному код.

Тестирование системных ограничений включает широкий набор относительно самостоятельных тестов, которые относятся к таким вопросам, как пользовательский интерфейс, базы данных, авторизация, производительность, утяжеленный режим, поведение при отказах, конфигурация и инсталляция приложения. Некоторые из тестов системных ограничений проводятся в параллель с тестированием системных услуг, другие выполняются независимо.

Тестирование и управление изменениями требуют разработки специальной документации, такой как документы по планированию тестов, документы по тестовым прецедентам, документы по дефектам и усовершенствованиям. *Тестовые требования* идентифицируются в *документах по тестовым прецедентам* и увязываются с документами по требованиям-прецедентам и прецедентам. Запрос на изменение обычно связан с дефектом или усовершенствованием.

Средства управления изменениями позволяют отправлять запросы на изменения и отслеживать их продвижение после обращения к ним разработчика. Важная часть средств управления изменениями связана с установлением путей прослеживаемости между запросами на изменения и другими системными артефактами, в частности, тестовыми требованиями и требованиями-прецедентами.

Вопросы

B1. В чем отличие сквозного контроля от инспекции?

B2. В чем заключается роль группы обеспечения качества (SQA-группы) в организации?

B3. Что такое база данных авторизации? Какова ее роль в разработке и тестировании системы?

B4. Какие виды тестирования системных ограничений тесно связаны с тестированием в утяжеленном режиме? Поясните свой ответ.

B5. Какие виды тестирования системных ограничений тесно связаны с инсталляционным тестированием? Поясните свой ответ.

B6. Какие действия возможны в ответ на отправленный запрос на изменения?

B7. Что такое “подозрительная траектория”? Приведите пример.

B8. Что такое точка верификации?

B9. Объясните различие между файлом базисных данных и фактическими данными.