В. В. Оносовский, А. Н. Терехов

ОРГАНИЗАЦИЯ РАБОТ В ПРОЕКТЕ RESCUEWARE

Организационные и технологические аспекты проекта RescueWare

Работы над проектом RescueWare в нашем предприятии ТЕРКОМ были развернуты сразу же после образования фирмы Relativity Technologies в феврале 1997 года. Для Relativity это было стратегическое решение — полностью поручить разработку своего основного продукта, с которым связывались все надежды и долгосрочные перспективы компании, группе программистов из России, создавая для этого здесь полноценную организационную структуру. Принятие такого решения, по-видимому, было непростым для наших американских партнеров, учитывая как глобальные риски и сложности коммуникации, так и разницу в уровнях административно-управленческой культуры. С другой стороны, аргументами в нашу пользу были более чем трехлетний на тот момент успешный опыт сотрудничества с фирмой Seer Technologies (высшие менеджеры которой составили ядро Relativity) и сотрудничество нашего предприятия с другими западными партнерами, в частности с итальянской корпорацией Italtel. После начала работ перед руководителями как Relativity, так и ТЕРКОМа встала задача организации работ коллектива российских разработчиков по западным стандартам таким образом, чтобы разрабатываемый нами продукт мог выдержать жесткую конкуренцию в условиях американского рынка.

К моменту начала работ в нашем распоряжении были следующие ресурсы:

- две небольшие группы программистов (по 5-7 человек каждая) в Санкт-Петербурге и Новосибирске с опытом работ в области реинжиниринга для фирмы Seer Technologies;
- большая группа студентов и аспирантов Санкт-Петербургского государственного университета (порядка 20 человек), разрабатывавшая прототипы конвертеров из языка Кобол в языки C++ и Java, которые составили основу функциональности RescueWare;
- возможность привлечения новых разработчиков из числа студентов и аспирантов университета и сотрудников ТЕРКОМа.

Для работ по проекту RescueWare были организованы два коллектива — в Санкт-Петербурге (численностью около 40 человек, из которых 30 человек — собственно разработчики, а 10 — группа обеспечения качества) и Новосибирске (8 человек). Группа разработчиков состояла из нескольких подгрупп, занимавшихся разработкой различных подсистем продукта. Такая структура (и соотношение численности сотрудников в разных группах) в общих чертах осталась неизменной до настоящего времени.

В развитии нашей организации за прошедшие три года можно выделить несколько этапов, каждый из которых соответствовал определенному уровню технологической и административной зрелости.

Основной задачей первого этапа было внедрение в практику работы элементарных норм технологической и управленческой дисциплины, которые позволили бы выпустить в запланированные сроки первую версию продукта. В этот период в практику были введены:

- Детальное планирование работ с использованием инструментальных средств (MS Project).
- Система еженедельной отчетности сотрудников о проделанной работе. В конце недели каждый сотрудник представляет краткий отчет (weekly report), содержащий сведения о том, что было сделано за прошедшую неделю, с какими трудностями и проблемами ему пришлось столкнуться и чем он планирует заниматься на следующей неделе. Отчеты руководителей групп разработчиков отражают те же моменты, но уже для деятельности всей группы в целом. Сопоставляя материалы отчетов с планами работ, можно видеть реальную картину и оценивать степень соответствия хода выполнения работ плановому графику.
- Детальное регламентирование процесса тестирования продукта сотрудниками группы обеспечения качества. Это имело для нас принципиальное значение как первый «живой» пример хорошо формализованного и регламентированного процесса.

С технологической точки зрения, основным требованием на начальном этапе было хранение всех исходных данных проекта (текстов программ, постоянных данных и документации) в

едином репозитории, обеспечивающем автоматический контроль версий. Мы остановимся на этом подробнее в разделе «Подготовка и выпуск версий продукта».

Наступление второго этапа было связано с появлением первых клиентов — пользователей RescueWare — и, следовательно, с необходимостью налаживания процесса регулярного выпуска официальных инсталляционных версий продукта. Было решено организовать эту работу в США, на площадке Relativity, причем версия должна была формироваться непосредственно из исходных текстов программ. Соответственно возникла задача создания второго репозитория исходных текстов в США и оперативного поддержания соответствия между двумя репозиториями. В результате увеличилась степень распределенности работ (возник третий центр активности), что потребовало расширения канала доступа через Интернет. Таким образом, важным побочным эффектом этого этапа стало развитие информационной сети организации, что открыло путь к более глубоким инфраструктурным изменениям.

Следующей важной вехой стало создание группой обеспечения качества общей базы ошибок с возможностью on-line доступа к ней из любого места (Санкт-Петербург, Новосибирск, США). Это позволило упорядочить процесс регистрации обнаруженных ошибок и их исправлений, а также облегчило общение QA с разработчиками.

Когда система RescueWare достигла определенного уровня качества при обеспечении приемлемого объема функциональности, появилась возможность упорядочить процесс выпуска версий продукта так, чтобы принципиально отличающиеся друг от друга версии выпускались в среднем дважды в год. Это задало внешний ритм всему процессу работы над продуктом и потребовало ряда организационных изменений. Так, при шестимесячном цикле выпуска версии время, отводимое на комплексное тестирование, оказалось одним из самых критических факторов. Чтобы снизить число ошибок интеграции продукта, была внедрена система автоматического (еженощного) построения текущей версии системы (nightly builds). Это потребовало наведения технологической дисциплины во всем цикле разработки и стало вторым по счету хорошо формализованным и регламентированным технологическим процессом.

Время прохождения полного цикла тестирования продукта является критическим фактором обеспечения качества, поэтому максимально возможная автоматизация этого процесса является одной из приоритетных задач. Работа над ней началась полтора года назад, и несмотря на явные успехи в ее решении (некоторые типы тестирования удалось почти полностью автоматизировать), она остается одной из наиболее приоритетных.

Основные направления улучшения деятельности организации в настоящее время связаны с попыткой переосмыслить нашу работу с более общих позиций — в терминах функционирующих и взаимодействующих друг с другом бизнес-процессов. Такой анализ дает возможность яснее увидеть «узкие места» и понять возможности повышения эффективности работы организации в целом.

Большая часть организационных и технологических нововведений внедрялись в практику нашей работы по требованиям американских партнеров, и далеко не всегда воспринимались нами с энтузиазмом. Так, например, мы поначалу активно возражали против требований размещения общей базы ошибок на сервере Relativity и работы с ней нашей группы QA в режиме On-Line, поскольку качество доступа по Интернету в тот момент было явно недостаточным. Нам казалось, что такая организация взаимодействия сажает нас «на иглу» Интернет-канала и что в российских условиях нужно строить инфраструктуру на основе более медленных и надежных коммуникаций — например, посредством электронной почты. Однако время показало, что общее развитие Интернет-каналов в ТЕРКОМе, стимулированное этими требованиями, подняло всю коммуникационную инфраструктуру предприятия на новый уровень и позволило заметно продвинуться в организации других автоматизированных процессов, в частности, процесса автоматизированной ночной сборки версий. Такая ситуация повторялась не раз, и, в конце концов, в значительной степени изменила наше отношение к предлагаемым организационным решениям.

Ниже мы подробно остановимся на организации двух наиболее важных, с нашей точки зрения, процессов — процесса подготовки и выпуска версий продукта и процесса обеспечения качества.

Подготовка и выпуск версий продукта

Процесс выпуска версий

Организация выпуска программных продуктов в виде последовательности выдаваемых пользователям версий давно является общепринятой практикой. Это позволяет производителям,

с одной стороны, оперативно реагировать на запросы пользователей и меняющиеся требования рынка и, с другой стороны, объединять в версиях все сделанные за некоторый период изменения и улучшения, выдавая их во внешний мир в виде логически законченных фрагментов новой функциональности. Для различных программных продуктов периодичность выпуска новых версий и степень различия между последовательными версиями могут сильно различаться: от чуть ли не еженедельных обновлений операционной системы Linux до выпуска новых версий раз в несколько лет, что характерно, например, для больших СУБД. Да и сами понятия версии (release), исправлений и дополнений к версии (patches, service packs) сильно зависят от организации процесса производства ПО в конкретной компании и могут означать существенно разные вещи для различных программных продуктов и фирм. Описываемая ниже схема выпуска версий в нашей организации отражает практику работы, сложившуюся за последние три года работы над продуктом RescueWare. Не претендуя на какую-либо универсальность, эта схема кажется нам достаточно типичной для работ объемом около 50 человеко-лет в год.

Основной единицей процесса является *версия*, под которой подразумевается выдаваемый «наружу» функционально завершенный вариант программного продукта, прошедший цикл тестирования и снабженный пользовательской документацией. Как правило, одновременно ведется работа только над одной версией, хотя бывают и исключения из этого правила.

Версии могут выдаваться во внешний мир на разных стадиях готовности. Так, *Альфа-версия* является чем-то вроде анонса будущей функциональности и может только обозначать общие контуры будущего продукта. Основное требование к степени работоспособности альфа-версии — она должна позволять продемонстрировать выборочные фрагменты новой функциональности на примерах, подготовленных разработчиками. *Бета-версия* — это уже функционально полный вариант продукта, в котором исправлена большая часть ошибок. Хотя в бета-версии могут оставаться известные ошибки, она уже вполне пригодна для практического использования. Как правило, бета-версии передаются узкому кругу опытных и квалифицированных пользователей, от которых, в свою очередь, ожидается информация о новых обнаруженных ими ошибках (бета-тестирование).

Версия общего назначения (General Availability или GA) представляет окончательный вариант продукта, в котором исправлено большинство найденных ошибок и описаны все ошибки, оставшиеся неисправленными (с вариантами их обходов при использовании продукта — Work Around). Такая версия является основной для распространения и передачи пользователям на весь период до выпуска следующей версии.

На основе выпущенной GA версии могут выпускаться *приложения к версии* (Add-On), реализующие специфическую функциональность, которая нужна не всем пользователям. В системе RescueWare, например, в виде таких приложений оформляются конвертеры со специфических языков, не входящие в основной комплект версии продукта. Понятно, что для работы приложений необходимо иметь установленную базовую версию продукта.

В том случае, если в процессе эксплуатации версии обнаружилась серьезная ошибка, исправление которой не может быть отложено до следующей версии, приходится выпускать *исправление* этой ошибки (fix). Такие исправления оформляются в виде пакета файлов, который устанавливается поверх основной версии, подменяя собой отдельные компоненты продукта.

Иногда возникает необходимость в исправлениях или доработках GA версии до выпуска следующей версии (например, по срочному пользовательскому запросу). В таких случаях измененные компоненты продукта собираются в *пакет изменений* (service pack), который, подобно исправлению ошибки, устанавливается поверх базовой версии, подменяя собой ее компоненты. В отличие от исправлений ошибок, выпуск которых является реакцией на замечания пользователей, подготовка и выпуск пакетов изменений обычно заранее планируется самими разработчиками.

Основу организации процесса разработки составляет основной цикл выпуска GA-версий, на фоне которого ведется разнообразная дополнительная активность — разработка и выпуск приложений (Add-Ons), пакетов изменений (Service packs) и т. д. В практике нашей работы версии выпускаются дважды в год, что позволяет сочетать оперативность в передаче новой функциональности пользователям с тщательностью подготовки каждой новой версии. Сразу после выпуска, очередная версия становится базовой и при необходимости выполнения дополнительных работ (например, разработка Add-On), все они делаются на основе именно этой версии, даже если следующая версия уже «на подходе».

Время, выделенное на подготовку версии (около полугода) распределяется приблизительно следующим образом: 1–1,5 месяца — согласование спецификаций и планов; 2–2,5 месяца — собственно разработка и автономная отладка компонент, и 2–2,5 месяца — комплексная отладка и передача версии в службу обеспечения качества (QA). Дополнительные компоненты, как

правило, разрабатываются по своим собственным планам без жесткой синхронизации с основным циклом выпуска версий.

Контроль версий исходных текстов

Дисциплина хранения и контроля версий всех исходных текстов, постоянных данных и документов, относящихся к продукту, является важнейшим фактором, обеспечивающим стабильность и эффективность коллективной разработки. В проекте RescueWare с самого начала обеспечение контроля версий было обязательным требованием заказчика. В качестве основного инструментального средства, позволяющего хранить все данные проекта в едином репозитории и осуществлять контроль за всеми вносимыми изменениями, был выбран Visual Source Safe фирмы Microsoft. Основная причина, по которой мы предпочли Visual Source Safe аналогичным продуктам других производителей (PVCS компании Synergex, CVS), была более тесная интеграция с системами программирования Visual C++ и Visual Basic в рамках продукта Visual Studio.

Внедрение контроля версий в общий процесс работы над проектом RescueWare происходило постепенно. Оглядываясь назад, можно выделить несколько ключевых этапов такого внедрения. Содержанием первого этапа, начало которого совпало с началом работ, была собственно организация репозитория проекта на общедоступном файловом сервере и установление в коллективе разработчиков дисциплины, исключающей хранение исходных данных проекта где-либо еще. Каждый день в начале работы разработчики должны брать (check-out) исходные тексты из репозитория на локальные диски своих компьютеров, а по окончании работы помещать их обратно (check-in). Механизмы разделения доступа к репозиторию, предоставляемые Visual Source Safe, гарантируют, что ни одно сделанное разработчиками изменение не будет потеряно, даже если один и тот же исходный текст в течение дня параллельно модифицировался двумя программистами.

После того как дисциплина работы программистов с Visual Source Safe достаточно устоялась, наступила очередь второго этапа — использования возможностей Visual Source Safe для поддержания корпоративных стандартов оформления исходных текстов. Требования по оформлению, среди прочего, определяют формат обязательных комментариев — заголовков программных модулей, содержащих сведения о назначении программной компоненты, ее авторе, дате разработки и истории вносимых в исходный текст изменений. Visual Source Safe имеет возможности автоматизированного контроля за соблюдением таких требований при помещении исходных текстов в репозиторий.

Следующим этапом внедрения контроля версий стала организация *распределенного* репозитория во всех трех центрах, где ведется работа над продуктом — в США, Санкт-Петербурге и Новосибирске. В каждом из этих трех мест были организованы собственные SourceSafe-репозитории, содержащие исходные данные для «своих» частей проекта. Для обеспечения удаленного On-Line доступа в «чужие» репозитории был использован продукт Source Off-Site одноименной компании.

Организация распределенного репозитория позволила организовать процесс ежедневной автоматической сборки версий продукта, описанный более подробно в следующем разделе. Основное направление дальнейшего внедрения контроля версий на текущий момент — организация тестового репозитория для нужд службы обеспечения качества (QA) на тех же самых принципах.

Процесс непрерывной сборки версий

При выпуске первых версий продукта RescueWare наиболее болезненным был момент перехода от автономной отладки отдельных компонент к комплексной отладке всего продукта. В этот момент выявлялись многочисленные несогласованности и нестыковки между различными частями продукта, которые зачастую не позволяли даже собрать все компоненты воедино. Требовалась кропотливая ручная работа по выявлению и аккуратному исправлению несогласованностей и ошибок в программных интерфейсах, занимавшая до полутора недель, в течение которых продукт был неработоспособен.

Для решения этой проблемы было предложено организовать автоматизированный процесс построения текущей версии продукта, запускаемый по ночам на специально выделенном сервере. Начав регулярно запускаться задолго до окончания разработки версии, автоматическая еженощная сборка выявляла бы значительную часть ошибок интеграции сразу же после их возникновения.

Однако собирать версию продукта, механически беря самые свежие версии исходных текстов, не имеет никакого смысла, поскольку тексты, находящиеся в текущей работе, могут быть «сырыми» и даже не компилироваться. Включение версии компоненты в процесс автоматического построения версии продукта должно быть осмысленным решением разработчика и делаться в выбранные им моменты. Таким образом, версия текстов, являющаяся «кандидатом» на включение в автоматическую сборку, должна быть отделена от текущей версии, в которую программисты вносят изменения.

Для реализации этого принципа средствами SourceSafe все исходные тексты были «расщеплены» на две ветви — *рабочую* (development branch) и *кандидатскую* (candidate branch). Содержание исходных текстов из рабочей ветви — внутреннее дело разработчика (хотя хорошим тоном считается помещать в репозиторий тексты, которые по крайней мере успешно компилируются). Содержание ветви-кандидата — это относительно устойчивые варианты текстов программных компонент, перемещаемые из рабочей версии после проверки их работоспособности. На основе кандидатской ветви автоматически строится *кандидат-версия продукта,* которая и является основным результатом процесса автоматической сборки.

Решение о перемещении текстов компонент из рабочей ветви в кандидатскую принимают руководители групп разработчиков на основе своего представления о степени готовности компонент. Поскольку функциональная полнота компоненты при этом не обязательна, каждая компонента в процессе разработки попадает в кандидатскую ветвь многократно, в моменты относительной стабильности.

Процесс автоматического построения кандидат-версии продукта, реализованный в нашей организации, состоит из нескольких фаз.

Первая фаза — загрузка исходных текстов из удаленных репозиториев при помощи средств On-Line доступа к ним (в нашем случае — Source Off-Site). Продолжительность этой фазы определяется объемом загружаемых исходных текстов и скоростью доступа к ним через Интернет и в нашем случае составляет порядка двух часов.

Вторая фаза — собственно сборка продукта. Для каждого проекта (в смысле Visual C++ или Visual Basic) запускается в пакетном режиме соответствующий компилятор. Состав проектов и режимы запуска компилятора управляются наборами командных файлов. Длительность этой фазы в основном определяется производительностью сервера, на котором выполняется сборка.

По результатам сборки система автоматически формирует текстовое сообщение и рассылает его по электронной почте всем заинтересованным лицам.

Третья фаза — фиксация результатов сборки в репозитории. В случае успеха, тексты из кандидатской ветви подготавливаются к отправке на другой сервер, формирующий инсталляционную версию продукта (в нашем случае этот сервер находится в США); в противном случае выполняется откат (Roll-back) к ближайшей стабильной версии.

Четвертая фаза — пересылка текстов на сервер, строящий инсталляцию. Мощности нашего сервера в это время используются для автоматического пропуска пакетов тестов, проверяющих только что построенную кандидат-версию продукта

Пятая фаза — построение инсталляционной версии продукта.

Шестая фаза — помещение построенной инсталляционной версии в общедоступное место (например, на FTP-сервер). В случаях, когда это связано с перекачкой файлов по FTP, данная фаза может занимать весьма значительное время.

Мы описали базовый процесс автоматического построения версии, однако никто не мешает запускать параллельно процессы построения разных версий продукта. Такая необходимость может возникнуть, например, в случае параллельной работы над текущей версией продукта и над пакетом изменений (service pack) к предыдущей GA-версии. В этом случае необходимо позаботиться о корректном разделении доступа к общим ресурсам, в первую очередь — к репозиторию исходных текстов. Такое разделение может обеспечиваться как посредством жесткого расписания запуска отдельных фаз процесса, так и при помощи специальных средств синхронизации (семафоров и т. п.).

Если собственно сборка продукта занимает не слишком много времени, весьма полезной становится возможность запуска *построения пробных версий* во внеурочное время (в течение рабочего дня) по инициативе разработчиков. При этом строится версия продукта, содержащая свежие версии локальных компонент и загруженные накануне ночью версии удаленных компонент. Основной смысл пробных версий — в возможности проверки корректности компоненты сразу же после ее перемещения из рабочей ветви в кандидатскую, не дожидаясь результатов ночного процесса. При наличии достаточных свободных ресурсов такая практика может быть рекомендована для снижения вероятности ошибок во время полной сборки продукта.

Служба обеспечения качества (Quality Assurance)

Ресурсы

Группа обеспечения качества (QA) существует в нашей организации с самого начала работ над проектом RescueWare. Созданная по инициативе американских заказчиков, эта группа составляет порядка четверти общей численности нашего коллектива. Выделение более четверти всех ресурсов на тестирование (а именно так вначале воспринимались нами задачи QA) — в первые месяцы такое с трудом укладывалось в голове. Однако по мере развертывания деятельности Relativity, оформления RescueWare как продукта и осознания истинных его масштабов и объемов реализованной в нем функциональности, наши оценки начали смещаться скорее в противоположном направлении — чем больше QA, тем более мы застрахованы от всевозможных неприятных «сюрпризов» с качеством выпускаемой версии. Так или иначе, но уже на протяжении более чем трех лет группа обеспечения качества является одним из стержней общего процесса создания программного обеспечения в нашей организации. За это время изменилось понимание задач и методов работы QA, и в чем-то даже собственно предмета ее деятельности, однако роль и влияние этой службы на остальные процессы нашей деятельности все это время только росли.

Наличие большой и развитой группы обеспечения качества в программистской организации — явление для российских фирм пока что абсолютно нетипичное. Однако и для западных компаний (в первую очередь американских), производящих программное обеспечение, на этот счет нет общих правил — наряду с фирмами, где QA составляет до половины численности персонала, существуют и такие, где в явном виде служба обеспечения качества отсутствует и ее функции распределяются между остальными частями организации. На наш взгляд, потребность в выделенной службе QA тем больше, чем выше «связность» как разрабатываемых программных продуктов, так и бизнес-процессов, формирующих деятельность организации. Большая часть проблем в сложной системе возникает «на стыках» между ее элементами, и чем больше функционально различных взаимодействующих между собой компонент содержит разрабатываемый продукт, тем более тщательные и формализованные процедуры требуются для поддержания его качества на должном уровне.

Тестирование

Проверка и тестирование выпускаемых версий программных продуктов были, есть и будут одними из основных задач службы обеспечения качества. Поначалу нам вообще казалось, что понятие QA как раз и означает «группа тестировщиков»; с течением времени понимание стратегических задач QA существенно изменилось, но постоянный и непрерывный процесс тестирования остается той основой, которая позволяет поддерживать должный уровень качества выпускаемых версий ПО.

Особенностью процесса тестирования является его достаточно хорошая формализуемость по сравнению, например, с процессом разработки. Как следствие, облегчается автоматизация различных его фаз. Опишем основные принципы организации процесса тестирования, сложившиеся в процессе работы над продуктом RescueWare.

Классификация ошибок

Целью тестирования программ является поиск ошибок. Понятие ошибки включает в себя множество совершенно различных ситуаций — от незначительных замечаний к дизайну пользовательского экранного интерфейса (форма и размер окон на экране, цветовая гамма и т. д.) до аварийного завершения программы при попытке выполнить важную пользовательскую функцию. Для классификации ошибок по степени критичности QA разделяет их на четыре уровня:

- 1-й уровень (авария) — ошибка, приводящая к аварийному завершению программы при выполнении пользователем разрешенной последовательности действий;
- 2-й уровень (ошибка функциональности) — ошибка, приводящая к неверным результатам при выполнении тех или иных функций программы;

- 3-й уровень (неудобство использования) — ситуация, приводящая к неудобству или неэффективности выполнения тех или иных функций программы, что снижает удобство работы с ней и привлекательность ее для пользователей;
- 4-й уровень (усовершенствование) — предложение по усовершенствованию реализации или интерфейса той или иной функции программы.

«Жизненный цикл» ошибки включает несколько стадий — от ее обнаружения и локализации, через работу над ней к ее исправлению разработчиками, проверке и включению этого исправления в очередную версию продукта. Соответственно можно выделить следующие стадии работы над ошибкой («статусы» ошибок):

- «Открытая ошибка» — обнаруженная ошибка, работа по исправлению которой еще не началась.
- «В работе» — данные об ошибке переданы разработчикам, которые занимаются ее исправлением.
- «Исправлена» — разработчики исправили ошибку, но это исправление еще не проверено QA.
- «Исправлена и проверена» — корректность исправления подтверждена QA.
- «Закрыта» — исправление включено в официальную версию продукта.

Опираясь на такую классификацию, можно формулировать четкие и ясные принципы соответствия версии продукта требованиям качества. Так, версия, содержащая ошибки 1-го уровня, ни при каких обстоятельствах не может быть выдана пользователям. Версия, содержащая ограниченное число ошибок 2-го уровня, может быть выпущена в качестве бета-версии, но не может рассматриваться как кандидат на официальный релиз.

Уровень серьезности ошибки и степень срочности ее исправления — не всегда одно и то же (хотя ошибки 1-го уровня всегда исправляются в приоритетном порядке). При подготовке версии продукта для конкретного клиента может оказаться, что для него принципиально важным является исправление всех известных на данный момент ошибок в какой-то выделенной группе функций, и не столь важно наличие ошибок даже 2-го уровня в остальной части функциональности, которой этот клиент не пользуется. Для отражения таких ситуаций используется специальный признак срочности, что позволяет формулировать такие требования к версии, как, например: «Должны быть исправлены все ошибки 1-го и 2-го уровней и все ошибки 3-го уровня с признаками срочности».

Единая база ошибок

Важным этапом организации процесса обеспечения качества явилась организация общей базы ошибок, размещенной на сервере Relativity (примерно через год после организации службы обеспечения качества — до этого поддерживалось несколько локальных баз). Оперативный доступ к единой базе из Санкт-Петербурга и Новосибирска обеспечивается через Интернет при помощи специального клиентского программного обеспечения. В базе содержится информация обо всех известных на данный момент ошибках, в том числе и закрытых. Запись о каждой ошибке содержит следующую информацию:

- уникальный номер ошибки;
- уровень серьезности ошибки;
- детальное описание ошибки, включая последовательность действий, необходимых для ее воспроизведения, и имя тестового примера;
- статус ошибки;
- предполагаемая компонента продукта, содержащая ошибку;
- имя сотрудника QA, обнаружившего ошибку;
- признак срочности.

Вся информация вносится в базу только сотрудниками QA. Номер ошибки в базе является уникальным идентификатором ошибки для всей организации. Так, например, типичной ситуацией в период перед выпуском версии, когда большинство разработчиков занято исправлением ошибок, являются еженедельные отчеты от разработчиков (weekly reports), состоящие исключительно из длинных списков номеров исправленных ошибок.

Приведенная структура записи об ошибке позволяет легко делать выборки по статусам ошибок, признаку срочности, компонентам продукта и т. д. В настоящее время в нашей организации сложилась такая практика, при которой руководители групп разработчиков сами,

без специальных напоминаний со стороны QA, отслеживают текущее состояние по своим частям продукта, делают выборки актуальных ошибок и распределяют их между разработчиками конкретных компонент.

Решение о физическом размещении базы на американском сервере создало для нас определенные проблемы с доступом через Интернет. К сожалению, канал из Санкт-Петербурга в США не всегда может обеспечить качество передачи данных, требуемое для работы клиентских программ в режиме on-line, и мы не раз сталкивались с ситуацией, когда в течение нескольких дней наша служба QA не могла внести в базу необходимые исправления. В настоящее время ведется работа по переносу базы на сервер в Санкт-Петербурге с поддержкой реплицированной read-only копии на сервере Relativity.

Процесс тестирования

В жизненном цикле каждой версии продукта наступает момент, когда активная разработка программ заканчивается и вариант продукта передается в QA. Процесс тестирования переданной версии включает несколько этапов:

- Предварительное тестирование («smoke test»), цель которого — убедиться, что основные функции программы выполняются без явных грубых ошибок. На такое тестирование отводится строго ограниченное время (например, половина рабочего дня), и по его результатам QA должен решить, есть ли смысл тестировать версию дальше или она должна быть «с порога» возвращена на доработку. Понятно, что передача в QA версии, не проходящей даже предварительного тестирования, должна рассматриваться как чрезвычайная ситуация со всеми вытекающими отсюда последствиями для разработчиков.
- Регресс-тестирование — следующий этап, позволяющий проверить, что качество переданной в QA версии не хуже, чем качество предыдущей версии. В случае обнаружения регресса версия также возвращается на доработку, причем исправление ошибок, приводящих к регрессу, является приоритетным.
- Полное тестирование — наиболее полный и содержательный этап, к которому допускается версия продукта, прошедшая два предыдущих этапа. В ходе полного тестирования проверяются все известные ошибки и обновляется их статус в базе, а также вносятся в базу сообщения о новых ошибках. Как правило, ошибки, обнаруженные на этом этапе, исправляются уже в следующей версии, за исключением случаев, когда необходимо срочное исправление, — например, если обнаружена ошибка, не позволяющая проверять дальше значительный кусок функциональности. Этот этап, наиболее трудоемкий и длительный, может занимать существенное время — одну-две недели.

Время и ресурсы, требуемые для выполнения каждой фазы процесса тестирования конкретного продукта, хорошо предсказуемы. Это позволяет для каждой выпускаемой версии заранее запланировать последовательность моментов передачи в QA очередных порций исправлений от разработчиков, причем от версии к версии временные рамки такого процесса меняются не слишком сильно.

Автоматизация тестирования

С самого начала работы QA остро встал вопрос о максимальной автоматизации процессов тестирования. Сейчас, по прошествии более чем трех лет, мы приобрели некоторый опыт, позволяющий нам оценивать реальные возможности и узкие места такой автоматизации.

Любая большая и сложная программная система содержит функциональные компоненты двух типов — *пакетные*, которые непосредственно реализуют основные функции, и *диалоговые*, обеспечивающие интерфейс пользователя с системой. Автоматизация тестирования пакетных компонент оказывается существенно проще, чем автоматизация проверки интерфейсных компонент. Как правило, такая проверка организуется следующим образом: в пакетном режиме запускается набор тестовых примеров, результаты прохождения которых записываются в файлы, и затем эти файлы сравниваются с заранее заготовленными образцами. Все расхождения заносятся в протокол, который затем изучается.

Такое тестирование требует значительных объемов вычислительных ресурсов: на пропуск больших пакетов тестов затрачивается много времени, а во многих случаях время прохождения

пакета является критичным (например, при предварительном тестировании, или в случаях, когда пакет тестов запускается на ночь).

Как нам кажется, решение заключается в организации системы распределенного выполнения тестов, при которой пакеты оформляются в виде набора отдельных задач, читающих исходные данные с общего тестового сервера и возвращающих обратно результаты пропуска тестов.

Такие задачи могут выполняться на любом свободном компьютере в вычислительной сети организации, имеющем достаточно свободных ресурсов (в частности, места на дисках). Имея планировщик таких задач и набор компьютеров, свободных в определенное время (например, машины разработчиков по ночам), можно добиться весьма высокой эффективности пропуска тестов.

Для пакетных компонент могут быть практически полностью автоматизированы предварительное тестирование и проверка на регресс, а в значительной части — и полный цикл тестирования.

Намного сложнее обстоит дело с диалоговыми компонентами. Несмотря на наличие специальных инструментальных средств, позволяющих создавать тесты для них в виде детальных протоколов-записей сеансов работы пользователя, фиксирующих все его действия — нажатия клавиш, перемещения и нажатия кнопок мыши и т. д., мы так и не смогли достичь сколько-либо существенного выигрыша в общей производительности по сравнению с ручным тестированием. Основная причина — большая трудоемкость создания таких тестов в сочетании с постоянными изменениями в самих экранных интерфейсах тестируемого продукта, в результате чего на поддержание тестов в актуальном состоянии уходит больше времени и сил, чем на тестирование продукта вручную.

Сквозной процесс обеспечения качества

Обеспечение правильного разделения функций и ответственности между разработчиками и службой QA, с одной стороны, и организация процесса их взаимодействия, с другой, являются ключевыми факторами обеспечения качества разработки и выпуска продукта. Эта задача видится нам как нахождение баланса между двумя крайними позициями, которые заключаются в следующем: либо QA выступает прежде всего как партнер разработчиков в тестировании незаконченной версии программного продукта, обеспечивая интенсивный поиск ошибок, пропуск больших пакетов тестов и помощь разработчикам в локализации ошибок, либо его основная функция — сертификация уже готового и оттестированного продукта и определение соответствия его качества стандартам (что-то вроде ОТК на советских предприятиях).

В соответствии с первым подходом, в QA могут передаваться незаконченные, промежуточные версии продукта. Это действительно сильно облегчает разработчикам жизнь на завершающих этапах разработки, но с другой стороны, такая позиция может поощрять безответственное отношение программистов к качеству своего программного кода, позволяя включать в тестируемую версию даже совсем «сырые» фрагменты, не прошедшие стадии автономной отладки.

Другой подход — рассматривать QA прежде всего как службу контроля и сертификации — типичен для американских компаний. Он порождает трудности на этапе комплексной отладки продукта, так как при проведении комплексного тестирования силами самих разработчиков трудно обеспечить достаточную степень «покрытия» тестами всего разнообразий функций продукта. Разработчик при тестировании программы склонен прежде всего проверять наиболее сложные и тонкие, с его точки зрения, части алгоритма, практически не уделяя внимания моментам, которые кажутся ему тривиальными. Еще один характерный момент — приверженность разработчиков рассуждениям типа «Эта функция аналогична другой, которую я уже проверил, и там все было нормально, — следовательно, здесь тоже все будет хорошо». Такой подход нередко приводит к ситуациям, когда после нескольких недель интенсивного тестирования разработчиками продукт «сваливается» в первые несколько минут после начала предварительного тестирования его QA, при попытке выполнить совсем простую и безобидную функцию.

В попытке найти баланс между этими двумя подходами мы предложили следующую схему организации сквозного процесса взаимодействия разработчиков и службы обеспечения качества в течение всего периода от начала разработки версии продукта до ее выпуска:

Шаг 1. *Спецификация.* Разработка каждой компоненты продукта начинается с подготовки разработчиками ее спецификации. В идеальном случае спецификация должна содержать исчерпывающее описание алгоритмов и программных интерфейсов компоненты, достаточное для корректного ее программирования. Однако реализация такого «чистого» подхода на

практике означала бы необходимость тратить на разработку и сопровождение документации больше времени и сил, чем на собственно программирование, и делала бы весьма болезненными любые отклонения от первоначальной постановки задачи. Для достижения реальных целей вполне достаточно спецификации, содержащей общее описание состава функций, принципов реализации и интерфейсов компоненты. Критерием приемлемости такой спецификации является, на наш взгляд, то, достаточно ли содержащихся в ней сведений о функционировании будущей компоненты для разработки схемы ее исчерпывающего тестирования. Так, например, для визуальной диалоговой программы в качестве спецификации может выступать граф-схема возможных действий пользователя (через окна, меню и т. д.) с краткими описаниями сути действий, соответствующих каждой вершине графа.

Подготовив спецификацию на разрабатываемую компоненту, разработчики передают ее в службу QA.

Шаг 2. *Разработка плана тестирования.* Получив от разработчиков спецификацию новой компоненты, специалисты QA пытаются составить на ее основе схему тестирования этой компоненты, обеспечивающую достаточно полную проверку ее работы. Если сведений, содержащихся в спецификации, для этого недостаточно, спецификация возвращается программистам для доработки. Так с самого начала разработки естественным путем обеспечивается должный уровень качества спецификации.

На практике, разумеется, разработка схемы тестирования ведется совместно специалистами QA и разработчиками (например, этим могут заниматься руководитель группы разработчиков компоненты и сотрудник QA, который будет отвечать за ее тестирование). Однако требование достаточности спецификации остается принципиальным, поскольку является объективным критерием качества ее подготовки разработчиками.

Схема тестирования представляет собой набор описаний ситуаций, для которых проверяется правильность функционирования программы, включая проверку всех функций для различных комбинаций параметров (как верных, так и неверных), случаи граничных значений и т. д. Хочется подчеркнуть, что это именно схема, формулируемая в терминах выполняемых функций и комбинаций параметров, а не перечень конкретных тестовых примеров. Ни о какой формальной полноте, обеспечивающей тестовое покрытие, при составлении схемы речь также не идет — схема составляется эмпирически на основе опыта QA и знания внутренней структуры и алгоритмов реализации будущей компоненты.

Удобным представлением схемы тестирования является таблица (в общем случае многомерная), в которой измерения соответствуют основным факторам, влияющим на поведение проверяемой компоненты, а клетки — различным комбинациям значений этих факторов. При этом далеко не все комбинации факторов могут соответствовать осмысленным тестовым ситуациям.

Закончив составление схемы тестирования, QA передает ее разработчикам, которые могут после этого начинать регулярную разработку и подготовку тестовых примеров для данной компоненты продукта.

Шаг 3. *Разработка и подготовка тестовых примеров.* Параллельно с собственно программированием компоненты происходит подготовка разработчиками набора тестовых примеров в соответствии с подготовленной QA схемой тестирования. Каждому отдельному элементу схемы должен соответствовать тестовый пример, проверяющий требуемую функцию компоненты. В некоторых случаях можно воспользоваться уже имеющимися примерами из библиотек тестов QA (например, комплекты тестов для транслятора, проверяющих грамматику языка программирования), в других случаях требуется написание специальных тестов. Так или иначе, но наполнение схемы тестирования конкретными тестовыми примерами является обязанностью разработчиков на данном этапе. По мере готовности тестовые примеры передаются на согласование и утверждение в QA. Подготовленный разработчиками и согласованный с QA полный комплект тестовых примеров становится официальным базовым набором тестов для проверки данной компоненты, а успешное его прохождение — формальным критерием готовности компоненты к передаче ее в QA.

В процессе разработки могут измениться первоначальные спецификации программы и, следовательно, методика ее проверки. В таком случае, разработчики меняют текст исходной спецификации и совместно с QA повторяют шаг 2. Поскольку изменения в постановке задачи обычно касаются отдельных функций программы, такое сопровождение спецификаций и планов тестирования вполне может выполняться в фоновом режиме, не нарушая основной последовательности шагов процесса.

Шаг 4. *Автономная отладка.* К моменту завершения программирования компоненты («Code Ready») на руках у разработчиков должен быть готовый и согласованный с QA набор тестов.

После этого может начаться этап *автономного тестирования* компоненты силами самих разработчиков, цель которого — добиться правильного выполнения всех тестовых примеров при работе компоненты в составе версии продукта. Эта фаза может продолжаться достаточно долго, поскольку часть тестов может не выполняться из-за ошибок или неготовности других компонент. Завершение этого этапа означает окончание разработки компоненты и готовность ее к комплексной отладке.

Шаг 5. *Комплексная отладка.* Успешное прохождение автономных тестов для всех компонент является формальным критерием для начала *комплексного тестирования* версии продукта при активном участии QA. Основной объем работ по тестированию и занесению информации об ошибках в базу данных проводится сотрудниками QA; разработчики занимаются локализацией и оперативным исправлением ошибок.

Комплексное тестирование проводится как на базовых наборах тестов, которые использовались при автономной отладке (что позволяет выявить ошибки интеграции компонент, их интерфейсов и т. д.), так и на наборах тестов большего объема, например, на реальных пользовательских приложениях. Основная задача QA на этом этапе — обеспечить высокую интенсивность тестирования версии и оперативно информировать разработчиков обо всех обнаруживаемых ошибках; основная задача разработчиков — быстро и аккуратно исправлять ошибки, не внося при этом новых ошибок. Продолжительность данного этапа определяется QA на основе динамики уровня качества версии продукта.

Шаг 6. *Передача версии в QA и сертификационное тестирование.* После того как QA принял решение о завершении этапа комплексной отладки (другими словами, о готовности версии продукта в целом), наступает заключительный этап — передача версии продукта в QA и подготовка к выпуску. При этом предполагается, что большая часть ошибок в продукте была обнаружена на этапе комплексной отладки и все, что можно было исправить в рамках выпускаемой версии, исправлено. Таким образом, на этом этапе QA выполняет следующие задачи:

- проводит повторный цикл полного тестирования версии (включая предварительное и регресс-тестирование);
- формирует перечень изменений в функциональности по сравнению с предыдущей версией;
- формирует список известных ошибок (не исправленных) и способов их «обхода» для пользователей;
- проверяет полноту и качество пользовательской документации.

Если в процессе тестирования обнаруживается новая ошибка, пропущенная на этапе комплексной отладки, QA принимает решение либо о ее исправлении в следующей версии (ограничившись для данной версии упоминанием ее в списке известных ошибок), либо о срочном исправлении разработчиками. В последнем случае разработчики делают «точечное» исправление, проверяют его в автономном режиме и после этого передают исправленные компоненты в QA, который повторяет для версии полный цикл сертификационного тестирования.

Схема, которую мы описали, является достаточно общей и оставляет за рамками обсуждения много важных вопросов — таких как включение в общий процесс этапов настройки/перенастройки/отладки шаблонов для автоматизации тестирования, организация внутри подразделений разработчиков собственной службы тестирования (development testing), соотношение этой службы с QA и многие другие проблемы. Тем не менее описанный подход к организации процесса, как нам кажется, способен, с одной стороны, обеспечить сквозной контроль качества на самых ранних этапах работы над версией продукта, и с другой — заметно сэкономить усилия на наиболее объемных и трудоемких заключительных этапах.