Опыт использования визуального представления информации для контроля качества в процессе создания программного обеспечения

Т.Н.Попова tnp@tercom.ru
А.Е.Тиунова aet@tercom.ru
Л.А.Кожарина luda@tercom.ru

Санкт-Петербургский государственный университет
198504, Университетский пр., 28
Санкт-Петербург, Россия

Аннотация

В данной статье рассматривается накопленный в коллективе разработчиков системы реинжиниринга RescueWare опыт использования различных отчетов для контроля качества разрабатываемых программных продуктов, мониторинга процессов разработки и выпуска программных продуктов, а также управления этими процессами.

Введение

Качество программного продукта является одним из важнейших факторов его коммерческого успеха. Качество продукта — понятие многогранное, объединяющее в себе множество составляющих. Это, в первую очередь, то, что видно пользователю — надежность, эффективность, удобство в использовании продукта, полнота и полезность документации. Это и внутреннее качество — ясность архитектуры, соответствие исходных текстов стандартам разработки. Коротко же понятие *качественный продукт* можно определить как *продукт, успешно и стабильно выполняющий задачи пользователя, для решения которых он предназначен.*

Естественно, любого производителя программных продуктов интересует ответ на вопрос “как сделать продукт качественным?”. Давно уже стало очевидно, что одних контролирующих мероприятий на заключительной стадии выпуска продукта недостаточно — необходимо организовать сквозной процесс контроля и обеспечения качества на всех стадиях разработки и выпуска продукта.

Задача обеспечения высокого качества весьма не проста, особенно для больших программных комплексов, к каким относится система реинжиниринга RescueWare, к тому же не имеющая “правильного” решения — процесс все время развивается “от меньшего к большему”, дополняется и видоизменяется в ответ на изменение внешних требований, появление новых задач перед компанией и т.п.

Целью данной статьи является представление накопленного нами опыта использования различных количественных характеристик и графических форм для оценки качества программного продукта, мониторинга, координации и корректировки процесса разработки и выпуска программного продукта, т.е. для обеспечения качества продукта непосредственно в процессе его создания.

1 RescueWare и процесс разработки

Опишем основные черты системы RescueWare и технологического процесса ее разработки, необходимые для дальнейшего изложения.

1.1 Система RescueWare

RescueWare представляет собой большой программный комплекс, предназначенный для решения задач анализа и реинжиниринга устаревших программных приложений.

RescueWare является семейством сходных продуктов — в качестве входных языков поддерживаются несколько диалектов языка COBOL, PL/I и Adabas/Natural, при этом при совместной установке они объединяются в единый продукт, поддерживающий все входные языки. Кроме “основных” языков программирования RescueWare поддерживает также несколько операционных

систем и менеджеров транзакций (CICS, IMS, Unisys) с соответствующими описаниями экранных форм (BMS, MFS, DPS), баз данных (SQL DDL, DMS) и языками управления заданиями (JCL, ECL).

В качестве выходной информации RescueWare предоставляет различные отчеты и диаграммы в нескольких форматах, генерирует тексты программ на Java и Visual Basic, описания схем баз данных в виде SQL DDL и XML, а также извлекает различные типы бизнес-правил в виде текстов программ на исходных языках.

Вся функциональность RescueWare делится на две группы: визуальная (FrontEnd) — анализирующие инструменты с развитым пользовательским интерфейсом (диаграммные визуализаторы, средства гипернавигации по исходным текстам программ и др.), и пакетная (BackEnd) — методы, которые управляются различными опциями, но самостоятельного пользовательского интерфейса не имеют (верификация исходных объектов, генерация текстов на целевых языках и т.п.). Кроме того, существует еще управляющая часть продукта (Core) — описание структуры репозитория, файлы опций, скрипты и т.д.

1.2 Процесс разработки

Разработка RescueWare ведется уже около 5 лет, в настоящее время объем исходного кода системы составляет около 1,5 млн. строк. Версии продукта выходят 2–3 раза в год.

Коллектив разработчиков состоит из трех групп, находящихся на значительном расстоянии друг от друга — в Санкт-Петербурге (Россия), Новосибирске (Россия) и Кэри (Северная Каролина, США). Самая большая группа находится в Санкт-Петербурге, в свою очередь она подразделяется на три подгруппы в соответствии с разрабатываемой функциональностью: анализаторы (BackEnd Parsing, сокращенно BEP), генерация (BackEnd Generation, BEG) и извлечение бизнес-правил (Business Rules Extraction, BRE).

Одновременно с развитием продукта в нашем коллективе разрабатывался и формировался процесс обеспечения и контроля его качества.

Каждая версия последовательно проходит стадии планирования, разработки, тестирования разработчиками, тестирования группой обеспечения качества (Quality Assurance, QA) и выпуска общедоступной версии (General Available, GA).

Тестирование (с момента передачи версии в QA до выпуска), занимающее до 1/3 общего времени работы над очередной версией продукта, в свою очередь, также проходит несколько последовательных этапов, каждый из которых решает определенную задачу. В нашей практике это начальный, основной, доводочный и заключительный этапы. Целью *начального* этапа является приемка версии на тестирование группой QA. Во время *основного* этапа осуществляется тщательное тестирование всей версии, ошибки интенсивно исправляются, и к концу этапа код версии можно практически заморозить. На *доводочном* этапе исправляется незначительное количество ошибок, с которыми версия выпущена быть не может, и проверяются только эти исправления. Целью *заключительного* этапа является сертификация и выпуск версии.

На каждой стадии работы над продуктом, а также для перехода от одной стадии к другой, существует набор требований, которым версия должна удовлетворять. Для обеспечения качества выпускаемого продукта необходимо организовать постоянный контроль за выполнением этих требований.

В нашей практике инструментами контроля являются:

- ежедневная сборка рабочих версий продукта, включающая сбор статистики;
- ежедневное автоматизированное тестирование рабочих версий продукта, включающее сбор статистики;
- ежедневные отчеты по базе данных ошибок, в том числе отслеживание динамики изменения количества ошибок;
- планы тестирования, позволяющие отслеживать состояние процесса тестирования.

Часть этих элементов процесса достаточно подробно описана в [3], мы будем касаться их здесь лишь по мере необходимости.

Failed: Blue (7.1.00.969) build 03:44 Spb

Func	Σ	BE	FE	Core	SPb	Nsk	Cary
Base	✔	✔	✔	✔	✔	✔	✔
Common	✔	✔	✔		✔	✔	✔
Cobol	✘	✔	✔	✘	✘	✔	
PL/I	✔	✔	✔	✔	✔	✔	
Natural	✘	✘	✔	✔	✘		
HPS	✔		✔				✔
Official	✘	✘	✔	✘	✘	✔	✔
Unofficial	✔	✔	✔		✔	✔	✔

Рис. 1: Отчет о результатах сборки версии

2 Контроль качества на стадии разработки

Рассмотрим более подробно использование визуальной информации на стадии разработки версии и тестирования ее разработчиками.

2.1 Ежедневная сборка версий

Первым шагом в налаживании системы качества явилась ежедневная автоматическая сборка официальной рабочей версии продукта с рассылкой результатов всем руководителям команды. Технические аспекты сборки версий в условиях распределенной разработки изложены в [4], мы же остановимся на основных организационных аспектах.

Результаты каждой официальной сборки представляют собой матрицу (рис. 1), столбцы и строки которой представляют разбиение всех строящихся компонент на группы, а ячейки отражают статус соответствующей группы компонент — успешно или нет. Мы используем разбиения по типу функционально-

сти (FrontEnd — FE, BackEnd — BE, Core), по организационной структуре команды (Санкт-Петербург — Spb, Новосибирск — Nsk, Кэри — Cary) и по принадлежности к элементам выпускаемого семейства программных продуктов. В нашем случае такими элементами являются общие компоненты (Base, Common) и языково-зависимые компоненты (COBOL, PL/I, Natural). Пустые ячейки означают, что компонент данного типа не существует. Итоговые ячейки (Official и Σ) содержат суммарный результат.

Данная матрица позволяет практически мгновенно оценить результаты сборки.

Срыв официальной сборки является событием чрезвычайным, поскольку отсутствие свежей версии не позволяет разработчикам эффективно отлаживать интеграционные аспекты, что особенно существенно в условиях, когда разработка распределена по нескольким городам. В период завершения тестирования версии срыв сборки часто означает перенос сроков выпуска продукта. Поэтому были предприняты организационные мероприятия, направленные на снижение количества неуспешных сборок версий.

Каждый такой случай фиксируется в базе данных с указанием группы разработчиков (в соответствии с организационной структурой команды), ответственной за неудачную сборку. “Виновная” группа определяется автоматически на основе списка распределения компонент, но ответственный за процесс сборки может вручную откорректировать эту информацию, например, если сбой произошел вследствие изменений интерфейса смежной компоненты. Также вручную отделяются технические причины — например, сбой локальной сети.

Штрафные очки суммируются в течение установленного срока (3 месяца), после чего публикуются среди всей команды, и подсчет начинается с нуля. При превышении допустимого порога сбоев какой-либо группой разработчиков к ней могут быть применены штрафные санкции.

Введение такой статистики вызвало повышение ответственности разработчиков при помещении исходных текстов в официальную рабочую версию и значительно снизило количество неудачных официальных сборок версий, поскольку никому не хочется быть “самым невнимательным разработчиком”.

Для того чтобы облегчить разработчикам проверку их изменений, в дневное время по их запросам проводятся тестовые сборки версий, статистика по которым, естественно, не ведется.

2.2 Автоматизированное тестирование

Автоматизация тестирования является одним из очевидных и распространенных средств повышения эффективности тестирования. Она является одним из приоритетных направлений работы и в нашем коллективе с первых дней существования группы QA. Уже на начальных этапах развития RescueWare стала очевидной необходимость и возможность автоматизации тестирования его пакетных компонент отдельно от интерфейса, и в самом продукте был запроектирован специальный код, позволяющий осуществлять запуск тестируемых методов из внешней программы.

Задача организации автоматизированного тестирования сама по себе сложна, требует тщательного продумывания и отдельных исследований. Для ознакомления с этими вопросами можно обратиться, например, к [2]. Мы остановимся здесь только на тех аспектах автоматизации тестирования, которые относятся к теме нашей статьи.

Первоначально автоматизация тестирования предназначалась для освобождения инженеров QA от повторения большого количества рутинных операций и обеспечения возможности протестировать пакетные компоненты продукта за один цикл тестирования, что вручную сделать практически невозможно. Очевидно, что это значительно повышает качество тестирования, а значит, и надежность конечного продукта, поскольку отсутствует риск при очередном повторении цикла тестирования пропустить какую-либо группу тестов. Кроме того, высвободившиеся при автоматизированном тестировании ресурсы можно использовать для углубленного тестирования других компонент продукта.

С развитием системы обеспечения качества к этим двум очевидным преимуществам, полученным от автоматизации тестирования, прибавились еще два — возможность быстрого получения объективной картины состояния каждой рабочей версии продукта и возможность использования результатов тестиро-

вания разработчиками непосредственно в процессе разработки, так как автоматизированное тестирование стало проводиться ежедневно в сочетании с автоматической сборкой версий.

Тем самым, автоматизация тестирования дала возможность перейти от тестирования конечного продукта к тестированию и обеспечению качества в процессе разработки, позволила превратить процесс тестирования из творческого в управляемый и повторяемый и сделала реальным получение информации о состоянии каждой рабочей версии.

2.2.1 Организация тестирования

Рассмотрим кратко схему организации автоматизированного тестирования.

Основной единицей тестирования является *тестовый проект* — набор тестовых файлов (пакет тестов) и опций для его обработки. Проекты различаются по типу использования — для первичной проверки состояния версии (Smoke), основного тестирования (Gold), расширенного тестирования и др.

Каждый тестовый проект проверяет определенную часть функциональности продукта, называемую в автоматизированном тестировании *целью тестирования*. Для каждой цели тестирования разрабатывается тестовый *сценарий*, состоящий из последовательности определенных *шагов*. Шаги могут быть как значимыми для результата тестирования, так и незначимыми — обычно это шаги, выполняющие некоторую подготовительную работу; значимость шага может зависеть от конкретного сценария.

Тестирование проекта состоит в последовательном выполнении шагов сценария, на каждом из которых обрабатываются все тестовые файлы. Критерием оценки результатов, в зависимости от специфики шага, может являться отсутствие ошибок, обнаруженных самим тестируемым методом, успешная компиляция сгенерированного кода целевым компилятором, сравнение полученных данных с образцами и т.п. Для конкретного тестового файла шаг может завершиться:

- успешно — статус Passed (успешно) или Warnings (с предупреждениями);

- неуспешно — статусы Failed (с ошибками), Crashed (некорректно завершился) или Suspended (был прерван программой тестирования по истечении допустимого тайм-аута);
- не выполнялся — статус Skipped (шаг пропущен для данного теста, так как один из предыдущих шагов завершился с ошибкой).

Вся информация о конфигурации тестовых проектов, целях тестирования и т.п. хранится в базе данных. Результаты выполнения тестов также заносятся в базу данных. Кроме того, во время тестирования на файловом сервере сохраняется дополнительная информация — различные логи, файлы ошибок, некоторые промежуточные результаты — для того, чтобы облегчить разработчикам выявление причин неудачного выполнения тестов.

Количественной характеристикой качества версии в рамках автоматизированного тестирования является процентное отношение объема успешно выполненных тестов ($Score$) ко всем запускавшимся ($TotalWeight$):

$$Quality = \frac{Score}{TotalWeight} \times 100\%$$

Для вычисления этой характеристики была разработана система весов — условных единиц оценки объема тестирования; с деталями подсчета весов и вычисления качества версии можно ознакомиться в [2].

2.2.2 Представление результатов тестирования

Поскольку результаты автоматизированного тестирования активно используются с самого начала работы над очередной версией, для обеспечения удобства доступа к ним был разработан Web-интерфейс. Основным принципом его построения является последовательная детализация результатов тестирования. Рассмотрим кратко основные страницы интерфейса.

Главная страница (рис. 2) отображает общую статистику по результатам автоматизированного тестирования выбранной версии. На экране представлены как суммарные итоги (Quality, Score), так и результаты по категориям тестирования (BEG,

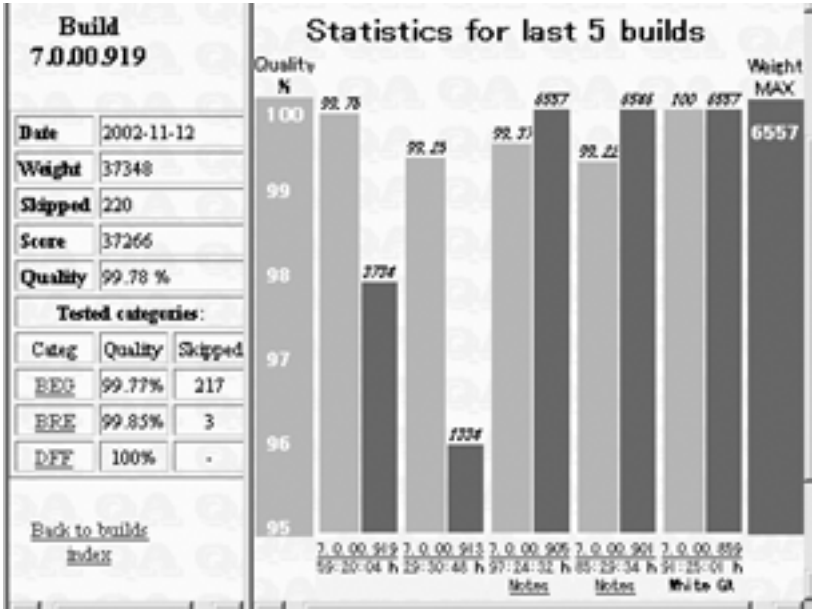


Рис. 2: Результаты автоматизированного тестирования: главная страница

BRE, DFF). Поскольку одним из важнейших направлений использования автоматизированного тестирования является регрессионное тестирование, большое внимание уделяется сравнению результатов с другими версиями. На главной странице размещена диаграмма сравнительной оценки качества и объема тестирования текущей версии (слева), 3-х предыдущих и последней из официально выпущенных версий, являющейся эталоном (справа). Каждая версия представлена двумя столбцами — качество и суммарный вес запускавшихся тестов. Наличие веса позволяет заметить, что по каким-то причинам тестирование могло быть проведено не полностью ($TotalWeight < MaxWeight$), и, соответственно, результат не отражает качество всего продукта.

На следующем уровне (рис. 3) представлена информация по категориям тестирования — на одной странице собрана информация обо всех проектах данной категории с указанием класса проекта, цели тестирования и использовавшегося набора опций. Сравнительные данные по другим версиям приведены в верхней

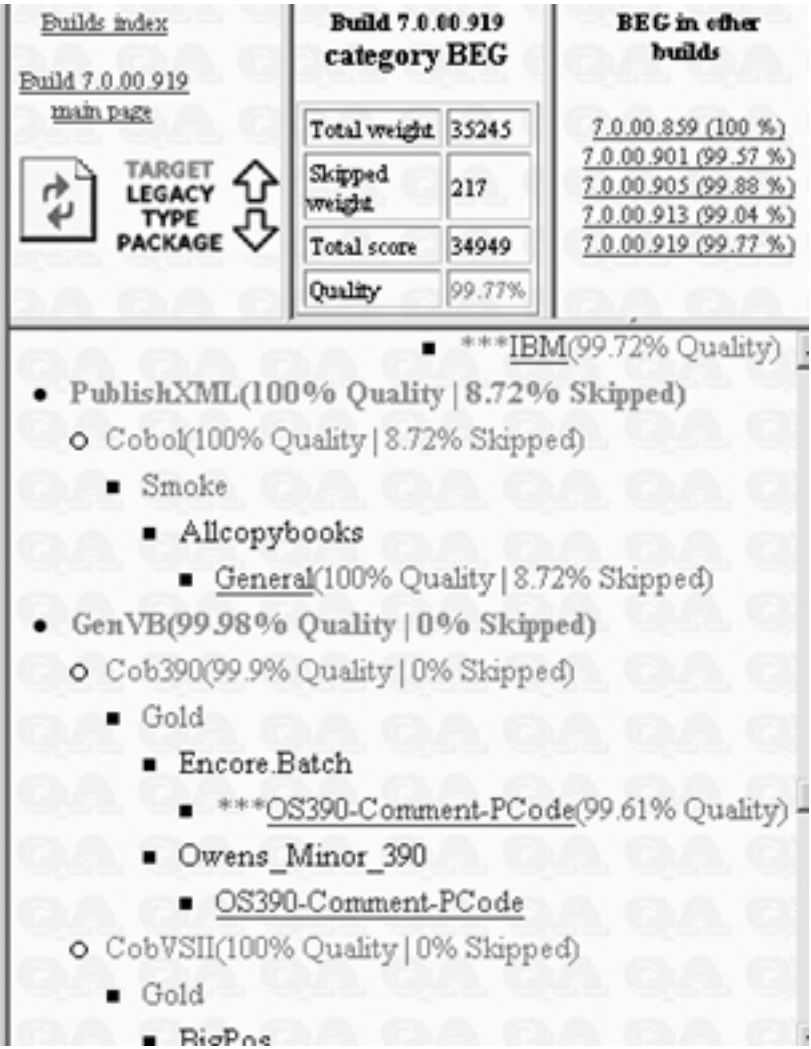


Рис. 3: Результаты автоматизированного тестирования по категории

части экрана (BEG in other builds); они имеют вид ссылок, по которым в новом окне браузера можно увидеть соответствующую страницу для выбранной версии. Также регресс и прогресс относительно предыдущей версии, как наиболее интересная для разработчиков информация, дополнительно показан цветом.

В зависимости от того, какая именно информация представляет наибольший интерес (особенно в процессе разработки, когда качество может быть далеко от эталонного), можно перегруппировать данные несколькими способами — по цели тестирования (Target), по классу проекта (Type) и т.п. с помощью кнопок, расположенных в верхнем левом углу. Как для каждого отдельного проекта, так и по группам проектов приведены промежуточные итоги.

Дальнейшие страницы подробно представляют результаты тестирования конкретного проекта вплоть до показа сообщений об ошибках, выявленных во время тестирования для каждого тестового файла в отдельности. На рисунке 4 представлена диаграмма, отображающая результаты тестирования отдельного проекта по каждому шагу сценария.

Кроме собственно результатов, на экране представляется общая информация о тестировании проекта и его опциях. Как обычно, для сравнения представлены результаты других версий для данного проекта.

Необходимым условием для передачи версии в QA является 100%-ное прохождение тестов класса Smoke, а для выпуска GA версии — тестов класса Gold. Представленная информация позволяет быстро проверить выполнение этих условий для текущей рабочей версии продукта.

3 Контроль качества на стадии тестирования

Перейдем к рассмотрению использования визуальной информации на стадии тестирования программного продукта группой обеспечения качества.

3.1 Отчеты по базе данных ошибок

За период работы над проектом RescueWare была разработана устойчивая и эффективная система работы с ошибками. Все

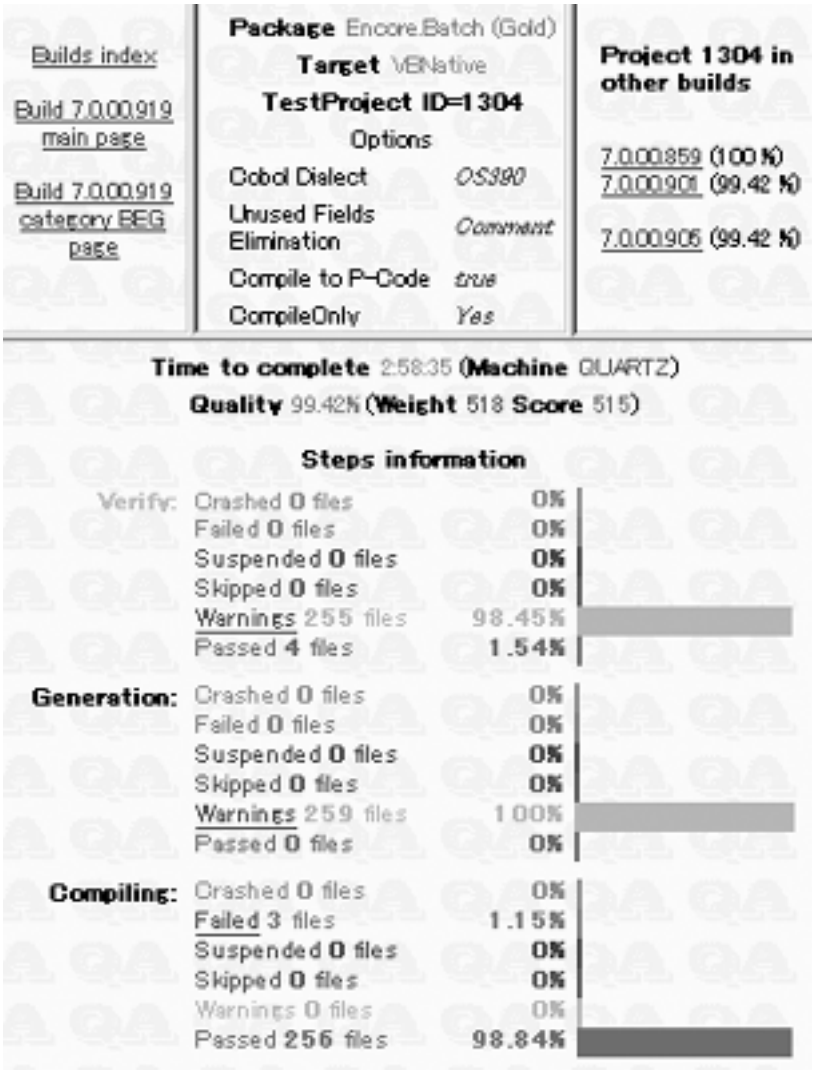


Рис. 4: Результаты автоматизированного тестирования по проекту

ошибки, обнаруженные при ручном тестировании, в обязательном порядке заносятся в базу данных, через которую и отслеживается история каждой ошибки.

С момента занесения ошибки в базу начинается ее жизненный цикл, в простейшем случае представляющий собой последовательную смену статусов:

- *Открыто* — присваивается инженером QA при обнаружении новой ошибки.
- *Исправлено* — изменяется разработчиком после исправления и самостоятельной проверки.
- *Закрыто* — присваивается после проверки исправления инженером QA.

На деле значительная часть ошибок проживает гораздо более трудную и интересную жизнь, в которой проблема может не воспроизвестись у разработчиков, быть отвергнута ими или объявлена дубликатом уже существующей, отложена руководителями до лучших времен или вновь воскрешена QA при тестировании новой версии. Вся ее биография прослеживается по изменению соответствующих этим событиям статусов и комментируется участниками в специальном поле базы данных.

Чтобы гарантировать, что в этом длинном и запутанном сериале ни одна из всех зафиксированных в системе ошибок не затерялась и не была забыта, необходимо регулярно формировать отчеты по всем имеющимся проблемам, включающие их описания, степень серьезности и т.п. Кроме этого, необходимо иметь сводную информацию о динамике изменения количества найденных в процессе тестирования и исправленных ошибок, чтобы видеть, стабилизируется ли версия, сможем ли мы выпустить ее в срок или, может быть, качество версии от цикла к циклу не улучшается и нужно предпринимать какие-то срочные меры для улучшения ситуации.

База данных ошибок в нашем случае создана на основе продукта Visual Intercept, предоставляющего для доступа как специальное клиентское программное обеспечение, так и Web-интерфейс. Для получения списков ошибок Visual Intercept позволяет создавать и выполнять запросы в интерактивном режиме, однако существенное время, затрачиваемое на прохождение

запроса из-за плохого качества связи (база данных находится у заказчика в США) и больших объемов общей и запрашиваемой информации, снижает эффективность работы. Кроме того, предоставляемые отчеты — это просто списки ошибок, и для того чтобы быть использованными в более широких целях, они требуют дополнительной постоянной обработки. С другой же стороны, работа по обработке информации базы данных для создания удобных и целенаправленных отчетов может быть легко автоматизирована, что и было нами сделано.

Система отслеживания ошибок была дополнена набором ежедневно автоматически обновляемых и рассылаемых всем участникам проекта электронных отчетов, подготовленных с помощью Microsoft Excel. Схема построения отчета проста:

- сначала вся необходимая информация загружается из Visual Intercept на листы Excel-книги с помощью ряда запросов, специфичных для каждого отчета. На каждом листе обычно размещаются ошибки, адресованные одной группе разработчиков;
- далее на отдельном листе строятся итоговые таблицы, суммирующие по определенным правилам всю информацию;
- наконец, итоговые таблицы отображаются на диаграммах.

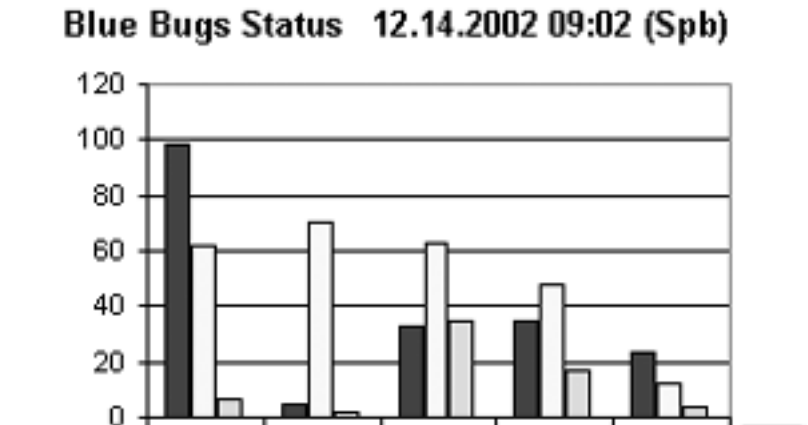
Наибольшую ценность в этих отчетах представляют диаграммы, хотя и наличие под рукой их исходных данных (списков ошибок) также часто оказывается полезным.

По своему типу отчеты можно подразделить на две группы:

- статические — отображающие текущее состояние качества продукта в фиксированный момент времени;
- динамические — отображающие изменение качества продукта во времени.

Примером статического отчета может служить Status Diagram (рис. 5).

Это наиболее востребованный отчет, особенно после начала тестирования очередной версии группой QA. Он содержит информацию обо всех имеющих отношение к данной версии ошибках, сгруппированных по статусам. К ним относятся все открытые на текущий момент ошибки (Open) и ошибки, исправленные

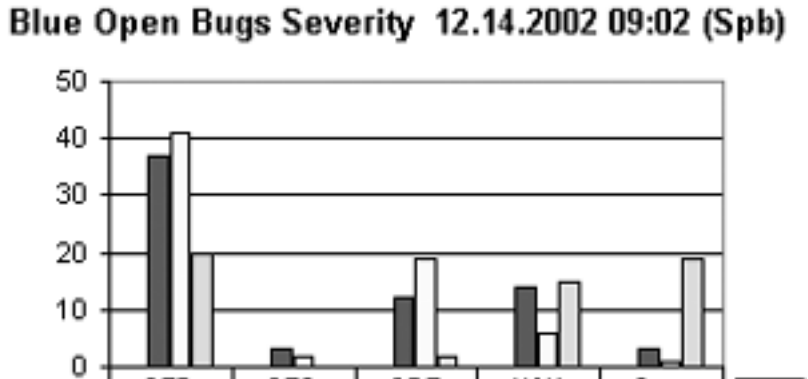


	BEP	BEG	BRE	NSK	Cary	Total
Open	98	5	33	35	23	194
Postpone	62	70	63	48	12	255
Fixed	7	2	35	17	4	65

Рис. 5: Отчет Status Diagram: распределение ошибок по статусам

в готовящейся версии продукта (Fixed). Графическая диаграмма позволяет произвести сравнительный анализ количества ошибок по их статусам в различных категориях (BEP, BEG, BRE, NSK, Cary). Особое внимание уделено открытым и отложенным для исправления ошибкам (Postponed), для них отдельно построены диаграммы, отображающие итоги по серьезности ошибок (рис. 6).

Для разработчиков полезными данными являются списки открытых ошибок в своей категории. Это позволяет им, например, выбрать наиболее удобную последовательность исправления ошибок, а также легко вычислить дублирующиеся ошибки. Инженерам QA необходимы списки ошибок, которые предстоит перепроверить, и списки ошибок с временными статусами (например Not Reproducible — ошибка не воспроизвелась у разработчика), которые также необходимо закрыть или открыть вновь после более тщательной проверки и уточнения. Отложенные проблемы должны находиться под постоянным контролем



	BEP	BEG	BRE	NSK	Cary	Total
Sev 1	0	0	0	0	0	0
Sev 2	37	3	12	14	3	69
Sev 3	41	2	19	6	1	69
Sev 0	20	0	2	15	19	56

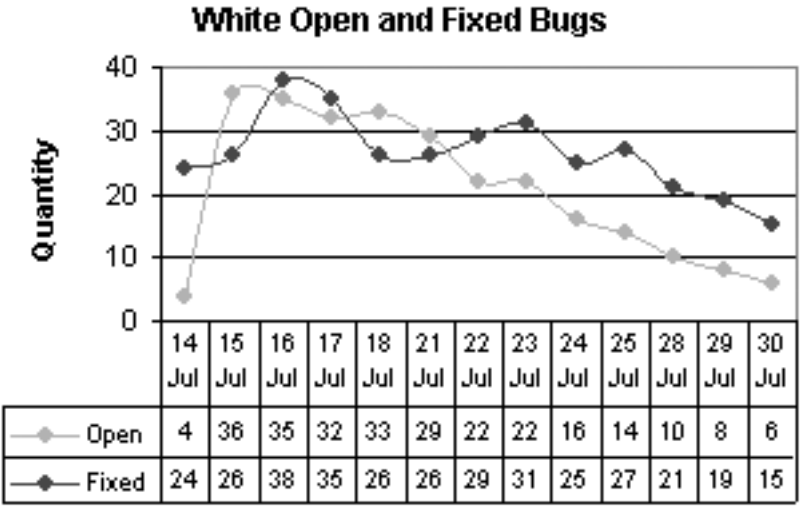
Рис. 6: Отчет Status Diagram: распределение ошибок по серьезности

руководителей и разработчиков. Руководство пользуется этим отчетом для отслеживания того, что важные ошибки не останутся неисправленными, помечая их как обязательные к исправлению в выпускаемой версии.

К статическим отчетам относятся также:

- Hot Issues, предоставляющий аналогичную информацию, но только по "горячим" ошибкам, без исправления которых версию выпустить нельзя. Этот отчет особенно актуален в период завершения тестирования, когда продукт "заморожен" и ошибки правятся только по специальному запросу;
- Customer Issues, специально предназначенный для работы с проблемами, обнаруженными пользователями продукта;
- Daily Status, отображающий итоги работы групп разработки и тестирования за день в виде количества найденных и исправленных ошибок.

Примером динамического отчета является History Report



	14 Jul	15 Jul	16 Jul	17 Jul	18 Jul	21 Jul	22 Jul	23 Jul	24 Jul	25 Jul	28 Jul	29 Jul	30 Jul
Open	4	36	35	32	33	29	22	22	16	14	10	8	6
Fixed	24	26	38	35	26	26	29	31	25	27	21	19	15

Рис. 7: Отчет History Report

(рис. 7). Этот отчет отражает не само качество продукта, а процесс его изменения. Списки содержат найденные и исправленные за предыдущий день ошибки. Итоговая таблица является накопительной по итогам каждого дня, а имеющиеся в отчете графики отображают динамику нахождения и исправления ошибок как для продукта в целом, так и для каждой категории в отдельности.

По графику четко прослеживаются тенденции роста количества найденных ошибок на начальном этапе тестирования версии (начиная с 15 июля для данного примера), последующего снижения (с 21 по 30 июля) в течение основного этапа тестирования и стабилизации на достаточно низком уровне (за пределами графика), позволяющей перейти к доводочному этапу. При этом количество ежедневно исправляемых ошибок находится на достаточно высоком уровне до окончания основного этапа тестирования.

На основании этого отчета можно оценить, укладываемся ли мы в график выпуска продукта. Опытным путем был выведен прогноз окончания работ над версией — если версия стабилизируется, т.е. в течение недели количество ежедневно обнаружи-

ваемых ошибок меньше некоторого фиксированного числа (скажем, 10) и при этом не превышает количество ежедневно исправляемых, а также не происходит резкого скачка количества ошибок в начале очередного цикла тестирования, то продукт может быть сдан в двухнедельный срок.

Weekly Report представляет собой укрупненный вариант предыдущего отчета. Итоговая таблица и соответствующие графики формируются по результатам недели, а не каждого дня. Этот отчет интересен с точки зрения возможности проведения сравнительного анализа хода работ для различных версий.

Все перечисленные выше отчеты архивируются после выпуска каждой версии, чтобы иметь возможность проведения сравнительного анализа и выявления тенденций и закономерностей за более длительные временные промежутки.

3.2 Тестовые планы

Хорошая тестовая документация выполняет три важнейшие функции [1]:

- облегчает тестирование;
- помогает организовать взаимодействие между персоналом;
- представляет собой удобную структуру для организации, планирования и управления тестовым проектом.

Как правило, тестовые планы рассматриваются только как технические документы и не применяются для управления работами по тестированию и отслеживания хода их выполнения. Между тем, тестирование продукта представляет собой самостоятельный проект, которым необходимо управлять. Для этого, прежде всего, нужна информация о состоянии процесса тестирования, на основании которой можно вовремя корректировать дальнейшие действия, учитывая уже имеющийся опыт, а также выявлять и решать возникающие проблемы по перераспределению ресурсов и уточнению сроков тестирования. С этой точки зрения электронный тестовый план может быть превращен в удобное средство управления проектом, позволяющее в любой момент получить отчет о ходе тестирования и объеме выполненных работ.

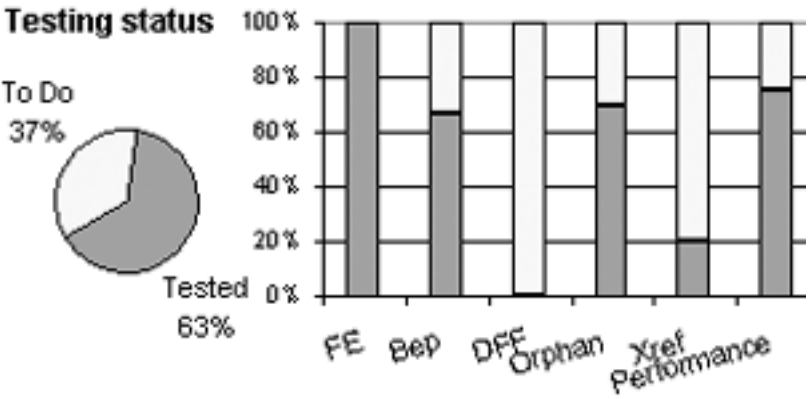


Рис. 8: Тестовый план: текущее состояние процесса тестирования

Тестовый план представляет собой набор таблиц, каждая из которых содержит план тестирования той или иной компоненты продукта. Для каждой компоненты сначала составляется список основных функций, после чего каждая функция разбивается на элементарные единицы тестирования. При тестировании визуальных компонент такими единицами могут быть пункты меню, для пакетных методов — конструкции входного языка. Для каждой единицы тестирования указывается необходимое число тестовых примеров, требующихся для ее проверки, а также способ выполнения тестов — вручную или автоматически.

Все единицы тестирования распределяются по категориям, определяющим время их выполнения и направление тестирования, к которым они относятся — первичная проверка (smoke test), основная функциональность, замеры производительности, проверка совместимости с предыдущими версиями продукта.

В процессе тестирования отметки о выполнении делаются непосредственно в тестовом плане, что позволяет легко отслеживать продвижение работ с возможностью представления итоговой информации в виде диаграмм.

Четкая структура тестового плана позволяет отслеживать разные аспекты процесса тестирования — как общее состояние, так и тестирование отдельных компонент или отдельных направлений, например, производительности продукта. Таким образом,

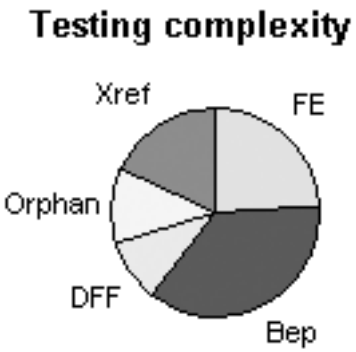


Рис. 9: Тестовый план: относительная сложность тестирования

практически в любой момент времени можно получить ответ на любимый вопрос руководства “на сколько процентов протестирован продукт?”, ответить на который без применения подобных средств затруднительно (рис. 8).

На основе тестового плана можно также сравнить относительную сложность тестирования различных частей продукта (рис. 9), что позволяет оптимальнее распределить ресурсы на тестирование, выявить те компоненты, автоматизировать тестирование которых необходимо в первую очередь.

Для компонент, тестирование которых уже полностью автоматизировано, тестовый план может являться планом дальнейшей автоматизации, т.е. включения тестов для проверки новой функциональности в пакеты автоматизированного тестирования.

Составление такого тестового плана делает процесс тестирования более систематизированным, легко повторяемым, позволяет убедиться, что никакая функция не упущена из виду, а также оценить трудоемкость тестирования всего продукта. Если по каким-либо причинам необходимо сократить время тестирования (провести неполное тестирование), то при наличии такого плана можно четко определить, какая функциональность продукта останется не протестированной и какой выигрыш во времени это даст.

При наличии тестовых планов последовательных версий продукта можно отслеживать динамику развития продукта (рост количества функций и возможностей), эффективность применения автоматизированного тестирования, изменение трудоемкости тестирования, а при наличии соответствующих графиков об ошибках выявить тенденцию изменения качества как компонент, передаваемых на тестирование, так и конечного продукта.

Заключение

Все описанные в данной статье способы представления информации о состоянии продукта и процесса его выпуска — таблицы, графики, статистические и "исторические" данные — появились и успешно используются в коллективе разработки системы реинжиниринга RescueWare. Это позволило, в сочетании с другими элементами управления, поднять на новый уровень эффективность работы коллектива и качество создаваемого продукта.

В ходе работы постепенно сформировались и основные принципы построения отчетов — четкая структура представляемой информации, наглядность представления и регулярность обновления.

Для больших программных комплексов нужны не просто абсолютные показатели — количество ошибок или процент выполнения плана, но и привязка этих показателей к отдельным категориям — по функциональности продукта или в соответствии с организационной структурой команды. Классификация (разделение продукта на несколько категорий по какому-либо признаку) нужна во всех аспектах деятельности, связанной с программным продуктом — при планировании, сборке версий, в тестировании и ведении базы ошибок, в рабочей и пользовательской документации, и чем меньше разнообразие используемых классификаций, тем удобнее работать. Два наиболее распространенных способа разбиения информации — это классификация по функциональности продукта и классификация в соответствии с организационной структурой команды. Заметим, что для небольших продуктов оба этих разбиения могут совпадать, но для больших систем, имеющих как развитый пользовательский интерфейс, так и мощные внутренние алгоритмы, это не так.

Очень важным моментом является наглядность представления имеющихся данных. Какой бы полезной ни была информация, если для ее анализа нужно сравнивать длинные столбцы цифр или выполнять какие-либо еще действия, то ценность информации уменьшается. Поэтому мы стремимся представить все необходимые данные в графической форме — в виде диаграмм и графиков различных видов.

Также необходимо обратить внимание на регулярность обновления и рассылки используемых отчетов всем участникам процесса, обеспечиваемую использованием автоматизированных средств. Если информация предоставляется от случая к случаю, то эффективность ее использования снижается.

Основной целью создания графических отчетов является представление в наглядной форме информации для отслеживания контрольных точек процесса и принятия решений. Они позволяют быстро и объективно оценить качество продукта, а также определить моменты перехода от одного этапа к последующему.

Если каждый отчет или график имеет четкую привязку к элементам общего процесса, то с ним легко связывается и набор конкретных регламентирующих действий. Например, отчет об имеющихся в версии ошибках служит для контроля требования “нельзя считать версию готовой к выпуску, если в ней есть ошибки наивысшего приоритета”, и все такие ошибки должны помечаться как обязательные к исправлению в текущей версии.

Система качества в коллективе постоянно совершенствуется, появляются новые и уточняются существующие ее элементы. В качестве следующего шага в развитии системы качества мы планируем создание специализированного Web-сайта в интранете, объединяющего всю информацию о процессе разработки и выпуска продукта, в том числе и все описанные выше отчеты и документы. Этот сайт должен стать интегрирующим звеном, облегчающим работу менеджеров проекта и разработчиков. Разработчики и группа автоматизации получат возможность в наглядной форме следить за процессом выполнения текущих задач (сборка, тестирование), а также инициировать выполнение новых задач. Руководство проектом сможет в одном месте получить всю необходимую информацию о выполнении плана, результатах тестирования и т.п. для составления оперативной, и

при этом объективной, оценки состояния процесса разработки и текущего качества продукта. Наличие такой объективной картины происходящего позволит более четко планировать работу коллектива.

Список литературы

[1] Канер С., Фолк Д., Нгуен Е.К. Тестирование программного обеспечения. — Киев: 2000. — 544 с.

[2] Комаров И.С. Автоматизация тестирования системы реинжиниринга RescueWare: Дипломная работа. — СПб.: 2000. — 37 с.

[3] Оносовский В.В., Терехов А.Н. Организация работ в проекте RescueWare // Автоматизированный реинжиниринг программ. — СПб.: 2000. — С. 43-63.

[4] Kiyaev V., Sobolev I., Terekhov A.A., Fedotov B. Formalization and Automation of Global Software Development Processes // Proc. of the 2nd International Workshop “New Models of Business: Managerial Aspects and Enabling Technology”. — 2002. — P. 150-159.