УДК 534.86

ТЕСТИРОВАНИЕ И КАЧЕСТВО ПРОГРАММНЫХ ПРОЕКТОВ В МЕТОДОЛОГИИ XP

О.Г. Старусев, Е.А Колмакова
(Национальный технический университет «ХПИ»)

Приведено доказательство необходимости тестировщика в XP-проекте.

тестирование, программный проект, методология XP

Актуальность. На сегодняшний день существует масса разнообразных методологий для разработки программного обеспечения (ПО). Одной из распространенных гибких методологий является методология XP (eXtreme Programming). Исходя из правил этой методологии, возникает вопрос о целесообразности совмещения функций тестировщика и разработчика в одном человеке.

Обзор проблемы. Extreme Programming – это концепция "агрессивного" и высокоэффективного процесса разработки программ. XP – это небольшой набор конкретных правил, позволяющих максимально эффективно выполнять требования современной теории управления программными проектами [1].

Основными правилами XP являются:

– написание User Stories – описание принципов работы системы;

– планирования Релиза. Релизы в XP частые и небольшие;

– проект делится на Итерации, каждая из которых начинается с собрания по планированию;

– персонал постоянно обменивается задачами;

– каждый день начинается с короткого собрания;

– разработка пробные решения (экспериментальные прототипы);

– новый функционал добавляются только по мере необходимости;

– «Безжалостный рефакторинг» – незамедлительное изменение архитектуры системы, или определенных ее частей при обнаружении неполадок. Мелкие ошибки игнорируются;

– заказчик постоянно присутствует в коллективе разработчиков;

– весь код должен соответствовать принятому стандарту;

– весь код должен быть создан «парным» программированием;

– частая интеграция кода;

– коллективное владение кодом (общий репозитарий проекта);

– оптимизация откладывается на более поздние итерации;

– если найдена ошибка, то тесты корректируются или создаются;

– функциональные тесты периодически выполняются и их результаты публикуются [2].

После подробного изучения процесса возникает вопрос: достаточно ли этого для создания действительно качественного программного обеспечения?

Цель работы. Доказательство необходимости тестировщика в XP проекте.

Описание предметной области. Для того чтобы ПО стало качественным, оно проходит отдельный жизненный цикл. Каскадная модель жизненного цикла процесса тестирования представлена на схеме (рис. 1).

Рис. 1. Каскадная модель жизненного цикла процесса тестирования

Таким образом, процесс тестирования является достаточно трудоемкой

и растянутой во времени фазой, а также требует особых знаний и умений.

Существуют две стратегии тестирования. Тестирование *"стеклянного ящика " (glass box)* – иногда ее еще называют *тестированием "белого ящика "* и тестирование *"черного ящика" (black box)*.

При тестировании "черного ящика" программа рассматривается как объект, внутренняя структура которого неизвестна. Проверяются только входные данные и правильность выходных данных без проверки правильности внутренней работы программы. Проводятся такие входные данные, которые могут привести к ошибочным или нестандартным выходным данным.

Как правило, наиболее часто проводимыми видами тестирования являются нижеперечисленные.

1. Функциональное тестирование, т.е. сопоставление выходных данных, полученных в ходе тестирования, с заданными в функциональных требованиях.

2. Стрессовое тестирование – тестирование продукта при номинальных или максимальных нагрузках с целью убедиться, что функциональность не вызывает сбоев, когда с системой работает большое число пользователей.

3. Регрессионное тестирование, проводимое после функционального усовершенствования или после исправления программы. Здесь ставятся две задачи:

а) убедиться, что обнаруженная ошибка исправлена и после этого не появились новые;

б) гарантировать функциональную преемственность и совместимость новой версии (релиза, патча) с предыдущими. Как правило, эта стадия включает повторное выполнение всех ранее использованных процедур тестирования.

4. Нагрузочное тестирование, или тестирование производительности – тестирование продукта при заданных известных нагрузках, которые определяются в терминах числа пользователей, объема данных и скорости их обработки (времени отклика).

Как видим, круг задач тестировщика не так уж мал, как принято считать. Но при "ручном" тестировании результаты каждого выполнения тестов пропадают, и их трудно повторить. Для увеличения объема проверок и повышения качества тестирования, обеспечения возможности повторного использования тестов при внесении изменений в ПО применяют средства автоматизации тестирования. К их числу относятся средства автоматизации функционального и нагрузочного тестирования клиент – серверных и Web-приложений: Rational TestStudio, Mercury LoadRunner, Segue SilkPerformer, а также менее популярные продукты

фирм Compuware, CA, Empirix, RadView Software и др.

Но следует учитывать, что на изучение и освоение этих средств автоматизации, будет потрачено не мало времени. Литературы на русском языке о перечисленных средствах на сегодняшний день не достаточно. Потому на автоматизацию будет потрачено достаточно времени и усилий.

Стоит ли требовать выполнения этой работы от разработчика, которому также необходимо следить за технологиями программирования, чтобы не потерять квалификацию разработчика?

Также не стоит забывать о тестовой документации, которая, как правило, является обязательным условием в успешном проекте. Одному человеку просто не хватит времени на то, чтобы охватить все эти моменты. К тому же придется очень часто переключаться с одной задачи на другую, что также не пойдет на пользу проекту.

Рассмотрим более подробно вторую стратегию тестирования – тестирование *"стеклянного ящика"*. В отличие от метода *"черного ящика"* данный метод основан на использовании определенных знаний программного кода, необходимых для контроля корректности данных на выходе. Тест является правильным только в том случае, когда известно, что конкретно должна делать программа. Значит, становится возможным контролировать ожидаемый результат. Тестирование методом *"белого ящика"* не обрабатывает случайные ошибки, но наряду с этим весь видимый код должен быть удобочитаемым. Например, это может быть отслеживание утечек памяти, целостности данных, тестирование модулей, проверка внутренней логики и др., т.е. компиляция и изменение программного кода в случае обнаружения ошибок.

Целесообразно для проведения тестирования “стеклянного ящика” обязанности по тестированию возложить именно на разработчика, так он как никто другой разбирается в собственном коде и сможет исправить собственные ошибки. Именно тестирование этих аспектов должны, на мой взгляд, охватывать Unit тесты методологии XP.

Следует также помнить, что тестирование принципиально отличается от программирования и по своим психологическим характеристикам. Программист, создавая код, порой на столько увлечен, что не может останавливаться, и проверять какие-то, на его взгляд, мелочи. Часто он откладывает решение мелкой проблемы «на потом» и занимается решением какой-то, более глобальной проблемой, после чего может напрочь забыть о первой. Но это вовсе не означает, что программист безответственно относится к разрабатываемой программе. Просто он решает более приоритетные вопросы. И он абсолютно прав. Во время создания программы, мысли программиста заняты не поиском и исправлением собственных ошибок. Он не может себе позволить отвлечься от спецификации.

Также следует учитывать тот факт, что относится объективно к ре-

зультатам своего труда очень тяжело, потому программист не сможет полноценно проверить написанный им же проект. Возлагать же на разработчика тестирование проекта, написанного другим разработчиком еще более не предусмотрительно, так как затраты на тестирование остаются те же что и в случае, если нанять профессионального тестировщика. Но, заставляя программиста занимаясь менее любимым делом - тестированием, существует риск лишиться разработчика.

Тестировщик же специалист с несколько иным назначением. Он способен рассматривать проблему с точки зрения пользователя, а обсуждать ее с разработчиком на языке программиста. Он является связующим звеном между разработчиком и конечным пользователем. Но основное отличие разработчика и тестировщика в том, что последний не чувствует угрызений совести, разрушая программный продукт. Что позволяет ему оставаться специалистом в области тестирования.

Выводы и перспективы. Таким образом, тестирование и программирование коренным образом отличаются друг от друга. Для тестирования требуется абсолютно иной склад характера, чем для программирования. Ведь задача тестировщика, доказать, что программа работает не правильно, а задача разработчика сделать программу такой, чтоб она максимально правильно работала. И независимо от того, какая методология применяется к проекту, один и тот же человек не сможет полностью и качественно совмещать в себе две эти задачи. Потому разрабатывать и тестировать ПО при использовании любой методологии должны разные роли.

ЛИТЕРАТУРА

1. *Бек К. Экстремальное программирование. – М.: ДМК, 2001. – 234 с.*
2. *Материалы форума «Экстремальное программирование в России». – [Электр. ресурс]. – Режим доступа: http://www.xprogramming.ru.*
3. *Джексон Л. Тестирование программного обеспечения. – М.: Солон, 2001.*
4. *Маейрс Г. Искусство тестирования программ. – М.: Финансы и статистика, 1982.– 288 с.*
5. *Байзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения. – С.-Пб.: Питер, 2004.*
6. *Элфилд Д. Автоматизированное тестирование ПО. – М.: ЛОРИ, 2003.*
7. *Материалы глобального форума «Экстремальное программирование». – [Электр. ресурс]. – Режим доступа: http://www.extremeprogramming.org.*
8. *Калбертсон Р., Браун К., Кобб Г. Быстрое тестирование. – М.: Вильямс, 2002.*
9. *Тамре Л. Тестирование программного обеспечения. – М.: Вильямс, 2003.*

Поступила 12.10.2005

Рецензент: доктор технических наук, профессор В.А. Краснобаев,

Харьковский университет Воздушных Сил.