ВВЕДЕНИЕ

В настоящее время, в связи с достаточно интенсивным развитием компьютерных технологий, все больше и чаще используются программное обеспечение (ПО), поэтому ошибка в работе этих программ может принести большие неудобства, затраты и даже убытки. В связи с этим, разработчикам ПО стоит уделять много времени и ресурсов тщательному тестированию, разрабатываемых ими программ. Качество программного обеспечения является постоянным объектом заботы при разработке и обсуждается во многих областях знаний.

В начале 1990-х годов в понятие «тестирование» стали включать планирование, проектирование, создание, поддержку и выполнение тестов и тестовых окружений, и это означало переход от тестирования к обеспечению качества, охватывающего весь цикл разработки программного обеспечения. В это время начинают появляться различные программные инструменты для поддержки процесса тестирования: более продвинутые среды для автоматизации с возможностью создания скриптов и генерации отчетов, системы управления тестами, ПО для проведения нагрузочного тестирования [1]. В середине 1990-х годов с развитием Интернета и разработкой большого количества веб-приложений особую популярность стало получать «гибкое тестирование» (по аналогии с гибкими методологиями программирования) [2].

1 МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

Модульное тестирование, или юнит-тестирование (англ. unit testing) − процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы [3].

Цель модульного тестирования заключается в изоляции отдельных частей программы, с целью впоследствии продемонстрировать работоспособность отдельных частей программы.

Этот тип тестирования обычно выполняется разработчиками (программистами). Идея его состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Таким образом, юнит-тестирование – это первый бастион на борьбе с ошибками (багами). За ним еще интеграционное, приемочное и, наконец, ручное тестирование, в том числе «свободный поиск» (рис. 1.1).

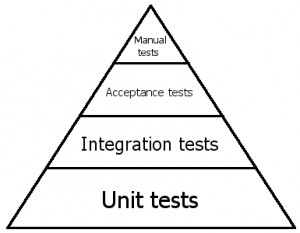


Рисунок 1.1 – Виды тестирования

1.1 Цели и преимущества модульного тестирования

*Поощрение изменений.* Модульное тестирование позже позволяет программистам проводить рефакторинг, будучи уверенными, что модуль по-прежнему работает корректно (регрессионное тестирование). Это поощряет программистов к изменениям кода, поскольку достаточно легко проверить, что код работает и после изменений.

*Упрощение интеграции.* Unit-тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх»: сначала тестируя отдельные части программы, а затем программу в целом.

*Документирование кода.* Модульные тесты можно рассматривать как «живой документ» для тестируемого класса. Клиенты, которые не знают, как использовать данный класс, могут использовать юнит-тест в качестве примера.

*Отделение интерфейса от реализации.* Поскольку некоторые классы могут использовать другие классы, тестирование отдельного класса часто распространяется на связанные с ним. Например, класс пользуется базой данных; в ходе написания теста программист обнаруживает, что тесту приходится взаимодействовать с базой. Это ошибка, поскольку тест не должен выходить за границу класса. В результате разработчик абстрагируется от соединения с базой данных и реализует этот интерфейс, используя свой собственный mock-объект. Это приводит к менее связанному коду, минимизируя зависимости в системе.

1.2 Техника модульного тестирования

Сложность написания модульных тестов зависит от самой организации кода. Сильное зацепление или большая зона ответственности отдельных сущностей (классы для объектно-ориентированных языков) могут усложнить тестирование [1]. Для объектов осуществляющих связь с внешним миром (сетевое взаимодействие, файловый ввод-вывод и т. д.) следует создавать заглушки. В терминологии выделяют более «продвинутые» заглушки – Mock-объекты, которые несут в себе логику. Также упростить тестирование может выделение как можно большей части логики в чистые функции. Они никак не взаимодействуют с внешним миром и их результат зависит только от входных параметров.

Код тестов принято выделять в отдельные каталоги. Желательно, чтобы добавление новых тестов в проекте не было сложной задачей и была возможность запускать все тесты. Некоторые системы контроля версий, например git, поддерживают хуки (англ. hook), с помощью которых можно настроить запуск всех тестов перед фиксированием изменением [1]. При ошибке в хотя бы одном из тестов, изменения зафиксированы не будут. Также можно применять системы непрерывной интеграции.

1.3 Стратегия модульного тестирования

Модульное тестирование является одной из ключевых практик методологии экстремального программирования (XP). Сторонники XP приводят следующие доводы в защиту этой практики [3]:

– написание тестов помогает войти в рабочий ритм;

– придает уверенность в работоспособности кода;

– дает запас прочности при дальнейшей интеграции или изменениях кода.

Ключевой фактор при оценке перспективности любого метода – стоимость проекта. Дополнительная работа по созданию тестов, их кодированию и проверке результатов вносит существенный вклад в общую стоимость проекта. И то, что продукт окажется более качественным не всегда перевешивает то, что он будет существенно дороже.

Известно, что продукт оптимальный по набору бюджет-функциональность-качество получается при применении различных способов обеспечения качества. Бездумное применение тотального модульного тестирования почти гарантированно приведет к получению неоптимального продукта. И никакие «запасы прочности» и «быстрый вход в рабочий ритм» не «спасут» проект от провала [2].

По мнению большинства разработчиков, модульное тестирование оправдано, если оно [3]:

– снижает время на отладку;

– дает возможность поиска ошибок с меньшими затратами, нежели при других подходах;

– дает возможность дешевого поиска ошибок при изменениях кода в дальнейшем.

Суммарный выигрыш от применения модульных тестов должен быть больше, чем затраты на их создание и поддержание в актуальном состоянии.

Отсюда следует, что цель модульного тестирования является получение работоспособного кода с наименьшими затратами. И его применение оправдано тогда и только тогда, когда оно дает больший эффект, нежели другие методы.

ПЕРЕЧЕНЬ ССЫЛОК

1. Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. — Киев: ДиаСофт, 2001. — 544 с. — ISBN 9667393879.

2. Лайза Криспин, Джанет Грегори. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд = Agile Testing: A Practical Guide for Testers and Agile Teams. − М.: «Вильямс», 2010. — 464 с. — (Addison-Wesley Signature Series). — 1000 экз. — ISBN 978-5-8459-1625-9.

3. http://software-testing.ru/library/testing/general-testing/77-2008-09-29-07-30-13