ВВЕДЕНИЕ

В настоящее время, в связи с достаточно интенсивным развитием компьютерных технологий, все больше и чаще используются программное обеспечение (ПО), поэтому ошибка в работе 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 процессе разработки и сопровождения программного обеспечения разработчикам приходится обращаться к различным видам динамических техник. В основном, это техники тестирования и отладки [1, 2].

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 [2].

1 КАЧЕСТВО ПРОГРАММНОГО ПРОДУКТА

Прежде всего, инженеры должны определить цели создания программного обеспечения. В этом контексте, особо важно помнить, что требования заказчика − первичны и содержат требования в отношении качества, а не только функциональности (функциональные требования). Таким образом, инженеры ответственны за извлечение требований к качеству, которые не всегда представлены явно, а также обсуждение их важности и степени сложности их достижения.

Все процессы, ассоциированные с качеством (например, сборка, проверка и повышение качества), должны проектироваться с учетом этих требований и несут на себе тяжесть дополнительных расходов (как важную составную часть стоимости программного обеспечения) [4].

Модель системы менеджмента качества представлена на рисунке 1.1.

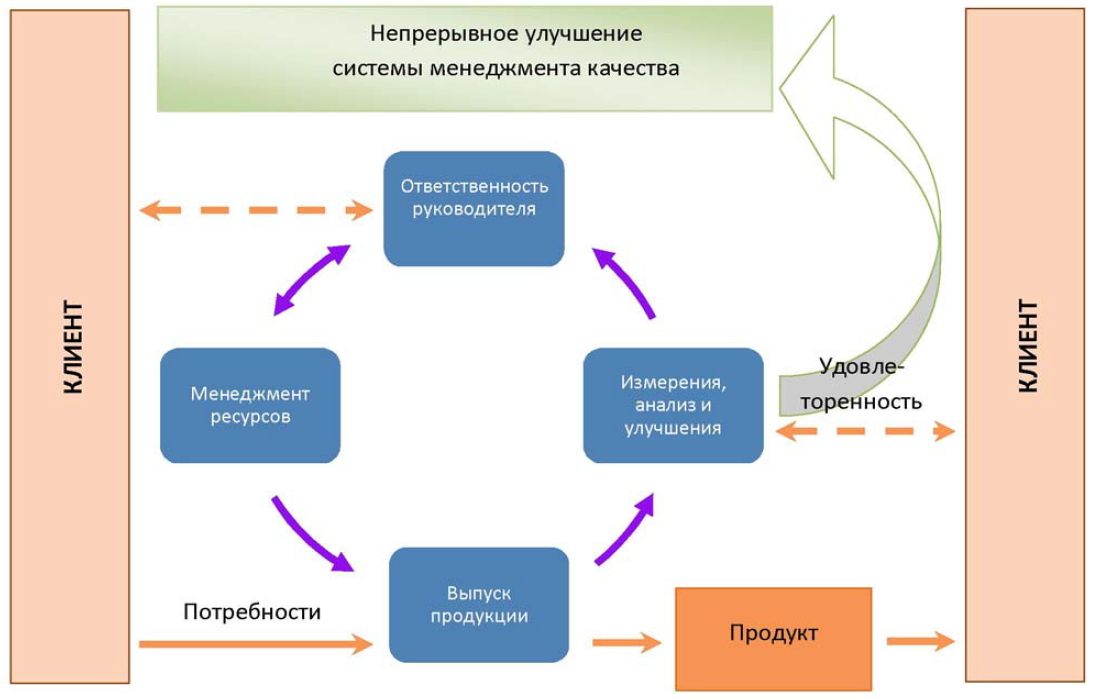


Рисунок 1.1 – Модель системы менеджмента качества

Хотя, чаще всего термин качество используется в отношении конечного продукта и поведения системы в процессе эксплуатации, хорошей инженерной практикой является требование к тому, чтобы соответствие заданным характеристикам качества оценивалось и для промежуточных результатов/продуктов жизненного цикла в рамках всех процессов программной инженерии.

1.1 Характеристики качества ПО

В качестве характеристик качества программного обеспечения выделяют следующие [1]:

– функциональность;

– надежность;

– удобство;

– эффективность;

– удобство сопровождения;

– портативность.

Функциональность (Functionality) – определяется способностью ПО решать задачи, которые соответствуют зафиксированным и предполагаемым потребностям пользователя, при заданных условиях использования ПО. Т.е. эта характеристика отвечает за то, что ПО работает исправно и точно, функционально совместимо, соответствует стандартам отрасли и защищено от несанкционированного доступа [1].

Надежность (Reliability) – способность ПО выполнять требуемые задачи в обозначенных условиях на протяжении заданного промежутка времени или указанное количество операций. Атрибуты данной характеристики – это завершенность и целостность всей системы, способность самостоятельно и корректно восстанавливаться после сбоев в работе, отказоустойчивость [1].

Удобство использования (Usability) – возможность легкого понимания, изучения, использования и привлекательности ПО для пользователя [3].

Эффективность (Efficiency) – способность ПО обеспечивать требуемый уровень производительности в соответствие с выделенными ресурсами, временем и другими обозначенными условиями [1].

Удобство сопровождения (Maintainability) – легкость, с которой ПО может анализироваться, тестироваться, изменяться для исправления дефектов, для реализации новых требований, для облегчения дальнейшего обслуживания и адаптироваться к имеющемуся окружению [1, 3].

Портативность (Portability) – характеризует ПО с точки зрения легкости его переноса из одного окружения (software/hardware) в другое [3].

На данный момент наиболее распространена и используется многоуровневая модель качества программного обеспечения, представленная в наборе стандартов ISO 9126. На верхнем уровне выделено 6 основных характеристик качества ПО, каждую из которых определяют набором атрибутов, имеющих соответствующие метрики для последующей оценки (рис. 1.2) [4].



Рисунок 1.2 – Модель качества программного обеспечения

1.2 Повышение качества

Качество программного обеспечения может повышаться за счет итеративного процесса постоянного улучшения. Это требует контроля, координации и обратной связи в процессе управления многими одновременно выполняемыми процессами [1]:

– процессами жизненного цикла;

– процессом обнаружения, устранения и предотвращения сбоев и дефектов;

– процессов улучшения качества.

К программной инженерии применимы теории и концепции, лежащие в основе совершенствования качества. Например, предотвращение и ранняя диагностика ошибок, постоянное совершенствование (continuous improvement) и внимание к требованиям заказчика (customer focus), составляющие принцип «building in quality». Эти концепции основываются на работах экспертов по качеству, пришедших к мнению, что качество продукта напрямую связано с качеством используемых для его создания процессов.

Конкретные работы и задачи по обеспечению качества структурируются с детализацией требований по их стоимости и ассоциированным ресурсам, целям с точки зрения управления и соответствующим расписанием в контексте целей, заданных планами управления, разработки и сопровождения [4].

Подходы к достижению качества таковы [5, 6]:

– качество достигается с помощью квалифицированных разработчиков, точного соблюдения процессов и удачных технологических подходов;

– качество достигается путем полного понимания всех действий и изменений; ни одна строка в программе не должна быть ни добавлена, ни изменена без полного понимания того что, зачем и как выполняется;

– качество достигается путем тщательного тестирования программы перед тем, как она будет доступна пользователю;

– достижение качества должно планироваться;

– достижение качества – обязанность каждого разработчика.

1.3 Тестирование и отладка

В процессе разработки и сопровождения программного обеспечения приходится обращаться к различным видам динамических техник. В основном, это техники тестирования. Однако, в качестве динамических техник могут рассматриваться техники симуляции, проверки моделей и «символического» исполнения (symbolic execution, часто предполагает использование модулей-«пустышек» с точки зрения выполняемой логики, с эмулируемым входом и выходом при рассмотрении общего сценария поведения многомодульных систем; иногда под этим термином понимаются и другие техники, в зависимости, от выбранного первоисточника) [4, 6].

Просмотр (чтение) кода обычно рассматривается как статическая техника, но опытный инженер может исполнять код непосредственно «в процессе» его чтения (например, используя диалоговые средства пошаговой отладки для ознакомления или оценки чужого кода).

Отладка ПО – деятельность, направленная на обнаружение и исправление ошибок в программе с использованием процессов выполнения его подпрограмм (модулей). Таким образом, отладку можно представить в виде многократного повторения трех процессов [6]:

– тестирования, в результате которого может быть констатировано наличие в программе ошибки;

– поиска места ошибки в программах и документации;

– редактирования программы и документации с целью устранения обнаруженной ошибки.

Успех отладки программы в значительной степени предопределяет рациональная организация тестирования. При отладке программы отыскиваются и устраняются, в основном, те ошибки, наличие которых устанавливается при тестировании. Тестирование не может доказать правильность программы [1], в лучшем случае оно может продемонстрировать наличие в нем ошибки. Другими словами, нельзя гарантировать, что тестированием программы практически выполнимым набором тестов можно установить наличие каждой имеющейся в ошибки.

Возникает две задачи [6]. Первая задача: подготовить такой набор тестов и применить к ним программное средство (ПС), чтобы обнаружить в нем по возможности большее число ошибок. Однако чем дольше продолжается процесс тестирования (и отладки в целом), тем большей становится продукта. Отсюда вторая задача: определить момент окончания отладки ПС (или отдельной его компоненты). Признаком возможности окончания отладки является полнота охвата пропущенными через ПС тестами (т.е. тестами, к которым применено ПС) множества различных ситуаций, возникающих при выполнении программ ПС, и относительно редкое проявление ошибок в программе на последнем отрезке процесса тестирования. Последнее определяется в соответствии с требуемой степенью надежности ПС, указанной в спецификации его качества.

1.4 Автономная отладка

При автономной отладке ПС каждый модуль на самом деле тестируется в некотором программном окружении, кроме случая, когда отлаживаемая программа состоит только из одного модуля. Таким образом, при автономной отладке тестируется всегда некоторая программа (тестируемая программа), построенная специально для тестирования отлаживаемого модуля. Эта программа лишь частично совпадает с отлаживаемой программой, кроме случая, когда отлаживается последний модуль отлаживаемой программы.

В процессе автономной отладки ПС производится наращивание тестируемой программы отлаженными модулями: при переходе к отладке следующего модуля в его программное окружение добавляется последний отлаженный модуль. Такой процесс наращивания программного окружения отлаженными модулями называется интеграцией программы [2]. Отладочные модули, входящие в окружение отлаживаемого модуля, зависят от порядка, в каком отлаживаются модули этой программы, от того, какой модуль отлаживается и, возможно, от того, какой тест будет пропускаться.

2 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).

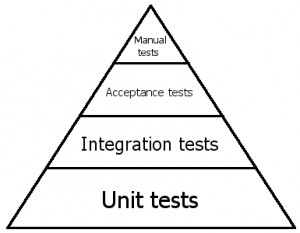


Рисунок 2.1 – Виды тестирования

2.1 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.

Таким образом, модульное тестирование мотивирует разработчиков писать код оптимизированным, проводить рефакторинг (упрощение кода программы, не затрагивая ее функциональность), так как с помощью Юнит-тестирования можно легко проверить работоспособность рассматриваемого компонента.

Также, при «разработке через тестирование», которая активно используется в экстремальном программировании, модульное тестирования является одним из основных инструментов, позволяющий разрабатывать модули в соответствии с требованиями к данному модулю.

2.2 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.

2.3 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 Кем К. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. / Канер Кем, Фолк Джек, Нгуен Енг Кек –Киев: ДиаСофт, 2001. – 544 с.

2. Криспин Л. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд / Лайза Криспин, Джанет Грегори − Москва: «Вильямс», 2010. − 464 с.

3. Модульное тестирование [Электронный ресурс] – Режим доступа: www / URL: http://software-testing.ru/library/testing/general-testing/77-2008-09-29-07-30-13 – 10.06.2017 г. – Загл. с экрана.

4. Качество программного обеспечения (Software Quality) [Электронный ресурс] – Режим доступа: www / URL: http://iiba.ru/software-quality/ – 10.06.2017 г. – Загл. с экрана.

5. Качество программного обеспечения [Электронный ресурс] – Режим доступа: www / URL: http://bourabai.ru/alg/system37.htm – 15.06.2017 г. – Загл. с экрана.

6. Кем К. Тестирование программного обеспечения. / Канер Кем, Фолк, Нгуен 2000- Сэм Канер, Джек Фолк, Енг Кек Нгуен − Киев: ДиаСофт, 2000.