Тестирование ПО

Тестирование ПО

Виды тестирования ПО

Долгое время основным способом тестирования программного обеспечения было тестирование методом «черного ящика» – программе подавались некоторые данные на вход и проверялись результаты в надежде найти несоответствия. При этом, как именно работает программа, считается несущественным. Этот подход до сих пор является самым распространенным в повседневной практике. Наряду с этим подходом существуют методы тестирования, которые изучают внутреннее устройство программы (исходные тексты, модули и их структуры). Их обобщенно называют тестированием «белого ящика».

Тестирование ПО

Также существует тестирование по методу «серого ящика», когда имеется частичный доступ к исходному коду (например, сторонние компоненты или автоматизация функционального тестирования через пользовательский интерфейс).

В настоящее время в литературе приведена обширная классификация видов тестирования ПО. Рассмотрим одну из них по следующим признакам:

1) По знанию внутреннего устройства программы:

– тестирование по методу «черного ящика»;

– тестирование по методу «белого ящика»;

– тестирование по методу «серого ящика».

Тестирование ПО

2) По объекту тестирования:

– тестирование требований к программному продукту;

– тестирование исходного кода (например, обзоры кода);

– модульное тестирование (проверка отдельных программных модулей);

– интеграционное тестирование (проверка взаимодействия программных модулей);

– объектно-ориентированное тестирование (тестирование классов);

– функциональное тестирование (проверка работы заявленной функциональности);

Тестирование ПО

– системное тестирование (проверка работы программы в реальном системном окружении);

– тестирование интерфейса программы;

– тестирование удобства использования;

– локализационное тестирование (проверка работы программы при переходах на другие иностранные языки);

– тестирование производительности;

– тестирование безопасности;

– тестирование на совместимость – проверка работы программы на разных операционных системах и в разных браузерах.

Тестирование ПО

3) По субъекту тестирования:

– тестирование, проводимое программистом (например, обзоры кода, модульное тестирование);

– тестирование, проводимое тестировщиком (например, функциональное тестирование, тестирование производительности);

– случайное тестирование (англ., ad hoc testing, проводится не участником проекта без предварительной подготовки с целью найти случайные ошибки в программе);

– приемочные испытания (проводятся, как правило, заказчиком).

Тестирование ПО

4) По позитивности тестов:

– позитивное тестирование;

– негативное тестирование.

5) По степени изолированности компонент:

– модульное тестирование;

– интеграционное тестирование;

– системное тестирование.

6) По степени автоматизации тестирования:

– ручное тестирование (проводится человеком);

– автоматизированное тестирование (полностью проводится компьютером);

Тестирование ПО

– полуавтоматизированное тестирование (проводится и компьютером, и человеком).

7) По степени подготовки к тестированию:

– тестирование по тестовым случаям;

– случайное тестирование.

8) По запуску программы на выполнение:

– статическое тестирование;

– динамическое тестирование;

9) По хронологии тестирования:

– до передачи пользователю (альфа-тестирование, тест приемки, тестирование новых функциональностей, регрессионное тестирование);

Тестирование ПО

– после передачи пользователю (бета-тестирование).

Альфа-тестирование – это тестирование ПО, когда процесс разработки приближается к завершению. В результате проведения такого тестирования в проект могут вноситься незначительные изменения.

Бета-тестирование означает тестирование, при котором разработка и тестирование по существу завершены, и до окончательного выпуска продукта необходимо обнаружить оставшиеся ошибки и проблемы.

Тестирование ПО

Экономическая сторона тестирования

Тестирование является затратной деятельностью, отнимающей время и деньги. Поэтому в большинстве случаев разработчики программного обеспечения заранее формулируют какой-либо критерий качества создаваемых программ (определяют так называемую планку качества), добиваются выполнения этого критерия и затем прекращают тестирование, выпуская продукт на рынок. Такая концепция получила название *Good Enough Quality* (достаточно хорошее ПО), в противовес более очевидной концепции *Best Possible Quality* (максимально качественное ПО).

Тестирование ПО

К сожалению, принцип Good Enough Quality зачастую понимают неправильно, ближе к формулировке Quality – If Time Permits (качество, если будет время). Конечно, выпуск плохо протестированного программного продукта из-за недостатка времени – это плохая практика. Опыт показывает, что пользователи склонны со временем забывать даже значительные задержки с выпуском продукта, но плохое качество выпущенного продукта запоминается на всю жизнь. На самом деле, Good Enough Quality – это просто поиск разумного компромисса между затратами на тестирование, длительностью разработки продукта и его качеством.

Тестирование ПО

Психология тестирования

Тестирование принципиально отличается от программирования по своим психологическим характеристикам. Дело в том, что программирование носит конструктивный характер, а тестирование деструктивно по своей природе. Можно сказать, что программист создает, строит, а тестировщик ищет недостатки в этих строениях. Поэтому программисты так часто не замечают очевидных ошибок в своих программах: к своему творчеству трудно относиться критично.

Принято считать, что команда разработчиков не должна совпадать с командой инженеров тестирования.

Тестирование ПО

Но при этом разработчики и тестировщики должны постоянно взаимодействовать друг с другом для достижения приемлемого качества окончательного продукта.

Модульное тестирование и его задачи

Модульное тестирование (англ., Unit Testing) – это вид тестовой деятельности, при котором проверке подвергаются внутренние рабочие части программы, элементы или модули независимо от способа их вызова. Под модулем принято понимать программу или ограниченную часть кода с одной точкой входа и одной точкой выхода, которая выполняет одну и только одну первичную функцию.

Тестирование ПО

Этот тип тестирования, как правило, осуществляется программистом, а не тестировщиком, поскольку для его проведения необходимы доскональные знания структуры и кода программы.

Методы, используемые при модульном тестировании, различаются по следующим признакам:

a) по степени автоматизации – ручные и автоматизированные методы;

b) по форме представления модуля – символьное представление (на языке программирования) или в машинном коде;

Тестирование ПО

c) по компонентам программы, на которые направлено тестирование, – структура программы или преобразование переменных;

d) по запуску программы – статические или динамические методы.

В первую очередь следует тестировать структуру программы, так как операторы анализа условий составляют в среднем 10–15 % от общего числа операторов программы.

Искажение логики работы программы приводит к серьезным ошибкам. К тому же данный вид тестирования имеет наилучшие показатели «эффективность/стоимость».

Тестирование ПО

Там же предлагается следующая методика тестирования модулей: последовательно проводить различные виды тестирования, начиная с самых простых:

– ручное тестирование (работа за столом);

– символическое тестирование (инспекции, сквозные просмотры);

– тестирование структуры;

– тестирование обработки данных;

– функциональное тестирование (сравнение со спецификацией, взаимодействие с другими модулями).

Тестирование ПО

Из приведенных выше видов тестирования, ручное и символическое относятся к статическому тестированию, которое проводится без запуска программы. Эти два вида, как правило, основаны на обзорах программного кода, только первый проводится автором-разработчиком, а второй – сторонними специалистами (другими разработчиками, инженерами по качеству или приглашенными инспекторами).

Последние три из перечисленных видов относятся к динамическому тестированию, и проведение каждого из них состоит из следующих этапов:

Тестирование ПО

– планирование тестирования (разработка тестов, формирование контрольных примеров);

– собственно тестирование;

– обработка результатов тестирования.

Обзоры программного кода

Обзоры программного кода являются основой статического тестирования и способны нейтрализовать действие человеческого фактора при выполнении поставленных задач.

При модульном тестировании можно выделить следующие положительные моменты обзоров (учитывая, что оно может проводиться как самим

Тестирование ПО

разработчиком, так и сторонними специалистами, которых будем называть экспертами):

1. Позволяют обнаружить посторонние элементы, что нельзя сделать в ходе традиционного тестирования. Обзоры могут следовать по всем выполняемым ветвям разрабатываемого ПО, а с помощью динамического тестирования невозможно проверить все ветви. В результате этого, редко проходимые ветви часто приводят к появлению ошибок.
2. Разные точки зрения, личностные качества и жизненный опыт помогают при обнаружении всех видов проблем, которые незаметны при поверхностном взгляде.

Тестирование ПО

3. Разработчики могут больше внимания уделять творческому аспекту процесса разработки, зная, что эксперты участвуют в проекте и действуют в качестве «подстраховки».
4. Происходит разделение труда, благодаря чему эксперты могут сконцентрироваться на этапе обнаружения проблем.
5. Распространение информации и обучение происходят тогда, когда разработчики встречаются при проведении обзоров кода. Люди быстро воспринимают и распространяют хорошие идеи, которые они замечают в работе других. По словам некоторых экспертов – это самый важный результат выполнения обзоров.

Тестирование ПО

6. Возрастает степень согласованности между членами команды. Происходит установление фактических групповых норм, что облегчает понимание сути программных продуктов и их сопровождение.
7. Менеджеры проекта могут получить представление о действительном состоянии разрабатываемых продуктов.

В конечном счете, в результате обзоров программные продукты улучшаются в силу следующих причин:

– Проблемы обнаруживаются тогда, когда их можно относительно легко исправить без значительных понесенных затрат.

Тестирование ПО

– Локализация проблем происходит практически в месте их возникновения.

– Инспектируются исходные данные какой-либо фазы с целью проверки их соответствия исходным требованиям или критериям выхода.

– Предотвращается возникновение дальнейших проблем с помощью оглашения решений часто происходящих затруднений.

– Распространяются сведения о проекте, что способствует повышению его управляемости.

– Разработчики обучаются тому, каким образом избежать возникновения дефектов при дальнейшей работе.

Тестирование ПО

– Предотвращается возникновение дефектов в текущем продукте, поскольку процесс подготовки материалов для инспекционной проверки способствует уточнению требований и проекта.

– Обучаются новые участники проекта.

– Менеджерам проекта предоставляются надежные опорные точки и предварительные оценки.

– Облегчается поддержка установленного порядка выполнения проекта и обеспечивается объективная, измеримая обратная связь.

Тестирование ПО

Тестирования структуры модулей

Тестирование структуры программного модуля происходит с помощью методов графического отображения модуля, одним из которых являются ориентированные графы МакКейба.

В качестве узлов в графах МакКейба выступают операторы, в качестве ребер – связи между операторами (рис. 14.1).

Тогда для каждого программного модуля может быть построен граф, который графически отобразит все возможные логические проходы по модулю. Проходом (в литературе также встречается термин «маршрут») будем называть путь от первой вершины графа до последней.

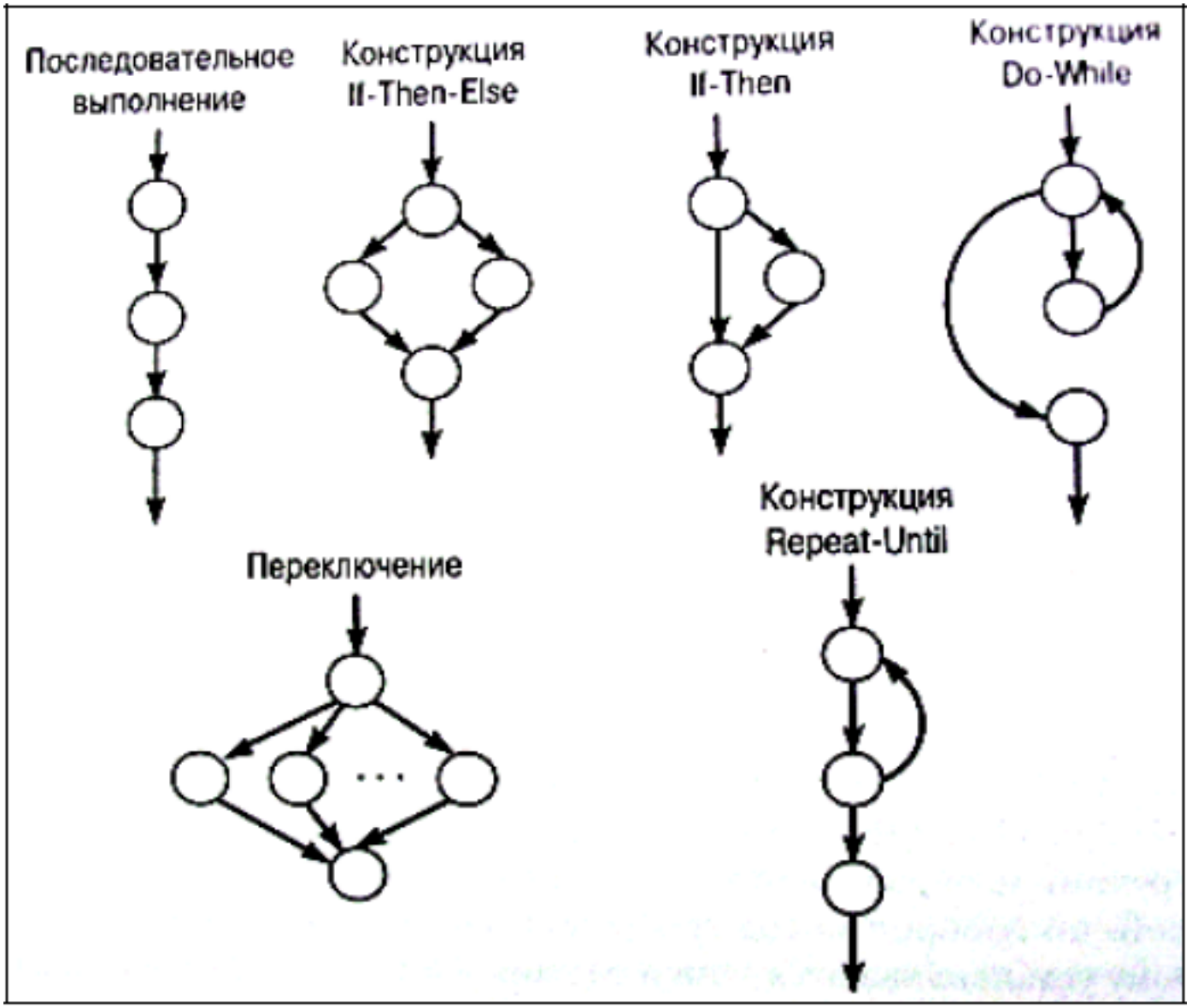

Рис. 14.1 - Графические конструкции МакКейба

Тестирование ПО

В основных графических конструкциях МакКейба, приведенных выше, отсутствуют некоторые современные конструкции, например, try-catch-finally. Для ее графического отображения можно воспользоваться конструкциями if-then или if-then-else.

При планировании тестирования структуры программных модулей решаются 2 задачи:

1) формирование критериев выделения маршрутов в программе;
2) выбор стратегии упорядочения выделенных маршрутов.

Тестирование ПО

Критерии выделения маршрутов в программе могут быть следующими:

a) минимальное покрытие графа программы;

b) маршруты, образующиеся при всех возможных комбинациях входящих дуг.

Стратегия упорядочения маршрутов выбирается по следующим параметрам:

– длительности исполнения и числу команд в маршрутах. Такая стратегия выбирается при тестировании программ вычислительного характера;

– количеству условных переходов, определяющих формирование данного маршрута.

Тестирование ПО

Такая стратегия выбирается при тестировании логических программ с небольшим объемом вычислений, а также в тех случаях, когда сложно оценить вероятность ветвления и количество исполнений циклов.

Тестирование взаимодействия модулей

После проведения модульного тестирования необходимо проверить совместную работу программных модулей, т. е. провести интеграционное тестирование.

Выделяют два способа проверки взаимодействия модулей:

Тестирование ПО

1) монолитное тестирование;
2) пошаговое тестирование.

Пусть имеется программа, состоящая из нескольких модулей, представленных на рис. 14.2. Главным модулем программы является модуль A, требуемые данные в программу попадают через модуль J, а выводятся на экран через модуль I.

Обозначения на рис. 14.2 следующие: прямоугольники – это программные модули, тонкие линии представляют иерархию управления (связи по управлению между модулями), жирные стрелки – ввод и вывод данных в программу.

Тестирование ПО

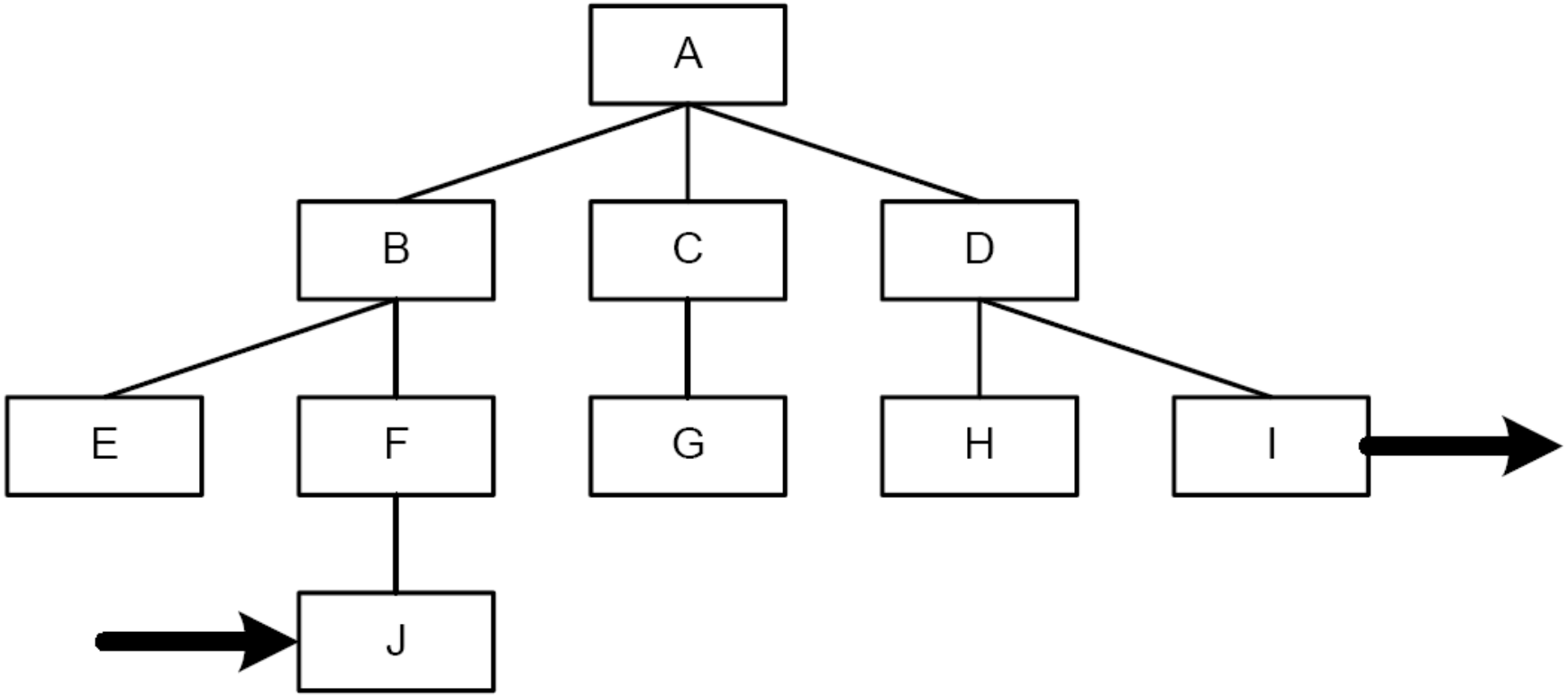

Рис. 14.2 - Схема многомодульной программы

Тестирование ПО

Монолитное тестирование заключается в том, что каждый модуль тестируется отдельно. Для каждого модуля пишется один модуль-драйвер, который передает тестируемому модулю управление, и один или несколько модулей-заглушек. Например, для модуля B нужны 2 заглушки, имитирующие работу модулей E и F. Когда все модули протестированы, они собираются вместе и тестируется вся программа целиком.

Пошаговое тестирование предполагает, что модули тестируются не изолированно, а подключаются поочередно к набору уже оттестированных модулей.

Тестирование ПО

Можно выделить следующие недостатки монолитного тестирования (перед пошаговым):

1. Требуется много дополнительных действий (написание драйверов и заглушек).
2. Поздно обнаруживаются ошибки в межмодульных взаимодействиях.
3. Следствие из 2 – труднее отлаживать программу.

К преимуществам монолитного тестирования можно отнести:

1. Экономию машинного времени (в настоящее время существенной экономии не наблюдается)
2. Возможность параллельной организации работ на начальной фазе тестирования..

Тестирование ПО

Стратегии пошагового тестирования

Существует две принципиально различные стратегии выполнения пошагового тестирования:

1) нисходящее тестирование;

2) восходящее тестирование.

Нисходящее тестирование начинается с главного модуля, в нашем случае с модуля А. Возникают проблемы: как передать тестовые данные в А, ведь ввод и вывод осуществляется в других модулях? Как передать в А несколько тестов?

Решение можно представить в следующем виде:

а) написать несколько вариантов заглушек модуля В (для каждого теста);

Тестирование ПО

б) написать заглушку B так, чтобы она читала тестовые данные из внешнего файла.

В качестве стратегии подключения модулей можно использовать одну из следующих:

– подключаются наиболее важные с точки зрения тестирования модули;

– подключаются модули, осуществляющие операции ввода/вывода (для того, чтобы обеспечивать тестовыми данными «внутренние» модули).

Нисходящее тестирование имеет ряд недостатков: предположим, что модули I и J уже подключены, и на следующем шаге тестирования заглушка H меняется на реальный модуль H.

Тестирование ПО

Как передать тестовые данные в H? Это нетривиальная задача, потому что между J и H имеются промежуточные модули, и может оказаться невозможным передать тестовые данные, соответствующие разработанным тестам. К тому же, достаточно сложно интерпретировать результаты тестирования H, так как между H и I также существуют промежуточные модули.

Восходящее тестирование практически полностью противоположно нисходящему тестированию. Начинается с терминальных (не вызывающих другие модули) модулей. А стратегия подключения новых модулей также основывается на степени критичности данного модуля в программе.

Тестирование ПО

Восходящее тестирование лишено недостатков нисходящего тестирования, однако имеет свой главный недостаток: рабочая программа не существует до тех пор, пока не добавлен последний (в нашем случае А) модуль.

Выбор одной из двух представленных стратегий определяется тем, на какие модули (верхнего или нижнего уровня) следует обратить внимание при тестировании в первую очередь.

Объектно-ориентированное тестирование

С традиционной точки зрения в модульном тестировании наименьшим элементом является элемент или модуль, который можно протестировать методом «белого ящика».

Тестирование ПО

При объектно-ориентированном тестировании этот функциональный элемент не отделяется от своего класса, в котором он инкапсулирован со своими методами. Это означает, что поэлементное тестирование по существу заменяется классовым тестированием. Однако не следует отклоняться слишком далеко от корневых принципов относительно того, что каждый метод класса можно рассматривать как «небольшой элемент», тестирование которого можно выполнять обособленно, применяя для этого методы «белого» и «черного» ящика. Объекты класса необходимо протестировать во всех возможных состояниях.

Тестирование ПО

Это означает, что необходимо сымитировать все события, вызывающие изменения состояния объекта.

В традиционном отношении элементы компилируются в подсистемы и подвергаются интеграционным тестам. При объектно-ориентированном подходе не применяется тестирование структурной схемы сверху вниз, поскольку точно не известно, какой класс будет вызван пользователем за предыдущим.

Интеграционные тесты заменяются тестированием функциональных возможностей, при котором тестируется набор классов, которые должны реагировать на один входной сигнал или системное

Тестирование ПО

событие, или же эксплуатационным тестированием, при котором описывается один способ применения системы, основанный на сценариях случаев эксплуатации.

Тестирование взаимодействия объектов следует по путям сообщения с целью отследить последовательность взаимодействий объектов, которая заканчивается только тогда, когда был вызван последний объект. При этом не посылается сообщение и не вызываются службы любого другого объекта.

Системное, альфа-, бета-тестирование и приемочное испытание пользователем изменяются незначительно, поскольку объектно-

Тестирование ПО

ориентированная система проходит эксплуатационные тесты точно так же, как и разработанные традиционным способом системы. Это означает, что процесс по существу не изменяется.

Тестирование классов охватывает виды деятельности, направленные на проверку соответствия реализации этого класса спецификации. Если реализация корректна, то каждый экземпляр этого класса ведет себя подобающим образом.

Тестирование классов аналогично тестированию модулей и может проводиться с помощью обзоров кода и выполнения тестовых случаев.

Тестирование ПО

Прежде чем приступать к тестированию класса, необходимо определить, тестировать его в автономном режиме, как модуль, или каким-то другим способом, как более крупный компонент системы.

Для этого необходимо выяснить:

– роль класса в системе, в частности, степень связанного с ним риска;

– сложность класса, измеряемая количеством состояний, операций и связей с другими классами;

– объем трудозатрат, связанных с разработкой тестового драйвера для тестирования этого класса.

Тестирование ПО

Если какой-либо класс должен стать частью некоторой библиотеки классов, целесообразно выполнять всестороннее тестирование классов, даже если затраты на разработку тестового драйвера окажутся высокими.

При тестировании классов можно выделить 5 оцениваемых факторов:

1. Кто выполняет тестирование. Как и при модульном тестировании, тестирование классов выполняет разработчик, поскольку он знаком со всеми подробностями программного кода. Основной недостаток того, что тестовые драйверы и программные коды разрабатываются одним и тем же персоналом, заключается в том, что неправильное

Тестирование ПО

понимание спецификации разработчиками распространяется на тестовые наборы и тестовые драйверы. Проблем такого рода можно избежать путем привлечения независимых тестировщиков или других разработчиков для ревизий программных кодов.

2. Что тестировать. Тестировать нужно программный код на точное соответствие его требованиям, т. е. в классе должно быть реализовано все запланированное и ничего лишнего. Если для конкретного класса характерны статические элементы, то их также необходимо тестировать. Такие элементы принадлежат самому классу, но не каждому экземпляру.

Тестирование ПО

3. Когда тестировать. Тестирование класса должно проводиться до того, как возникнет необходимость его использования. Необходимо также проводить регрессионное тестирование класса каждый раз, когда меняется его реализация. Однако до начала тестирования класса его необходимо закодировать и разработать тестовые случаи использования класса. Ранняя разработка тестовых случаев позволяет программисту лучше понять спецификацию класса и построить более успешную реализацию класса, которая пройдет все тестовые случаи.

   Существует даже практика первоначальной разработки тестовых случаев, а затем программного кода.

Тестирование ПО

Такой подход направлен на изначально все предусматривающий программный код и носит название разработка через тестирование (англ., Test-Driven Development, TDD).

Главное, чтобы этот подход не привел к проблемам во время интеграции этого класса в более крупную часть системы.

4. Каким образом тестировать. Тестирование классов обычно выполняется путем разработки тестового драйвера. Тестовый драйвер создает экземпляры классов и окружает их соответствующей средой, чтобы стал возможен прогон соответствующего тестового случая.

Тестирование ПО

Драйвер посылает одно или большее количество сообщений экземпляру класса (в соответствии с тестовым случаем), затем сверяет результат его работы с ожидаемым и составляет протокол о прохождении или не прохождении теста. В обязанности тестового драйвера обычно входит и удаление созданного им экземпляра.

5. В каких объемах тестировать. Адекватность может быть измерена полнотой охвата тестами спецификации и реализации класса, т. е. необходимо тестировать операции и переходы состояний в различных сочетаниях. Поскольку объекты находятся в одном из возможных состояний, эти состояния определяют значимость операций.

Тестирование ПО

Поэтому требуется определить, целесообразно ли проводить исчерпывающее тестирование. Если нет, то рекомендуется выполнить наиболее важные тестовые случаи, а менее важные выполнять выборочно.

Автоматизация модульного тестирования

Автоматизация модульного тестирования направлена за замену ручного запуска модульных тестов некоторым инструментальным средством. Для автоматизации модульного тестирования существует достаточно большое количество инструментов (их принято называть фреймворками), которое можно разделить на 3 группы:

Тестирование ПО

– семейство xUnit;

– встроенные фреймворки в среды разработок;

– универсальные инструменты для разных видов тестирования.

Семейство xUnit

Семейство xUnit – это набор программных средств для разработки модульных тестов, позволяющий реализовать построение тестов, их выполнение и вывод полученных результатов. Семейство xUnit имеет отдельную реализацию для большинства используемых в настоящее время языков программирования, например:

Тестирование ПО

– JUnit – для тестирования Java-кода;
– DUnit – для тестирования программ на Delphi;
– cppUnit – для тестирования c++ программ;
– NUnit – для тестирования .NET приложений;
– PyUnit – для тестирования программ, написанных на Python;
– VBUnit – для программ, написанных на VisualBasic;
– utPLSQL – для языка PLSQL (Oracle).

Также существует инструмент httpUnit, который реализован на языке Java и имеет следующие возможности: эмулирует основное поведение браузера, включая заполнения веб-форм, работу с элементами JavaScript,

Тестирование ПО

основную http-аутентификацию, автоматическую страничную переадресацию, поддерживает работу с cookies, позволяет тестовому коду проверить возвращаемую страницу в виде html-текста с контейнерами форм, таблиц и линков.

Используя возможности классов JUnit, позволяет легко создавать тестовые сценарии и быстро проводить автоматизированное тестирование функциональности веб-приложения.

Инструмент httpUnit позволяет работать напрямую с веб-сервером без участия веб-браузера, заменяя его собой (рис. 14.3–14.4).

Тестирование ПО

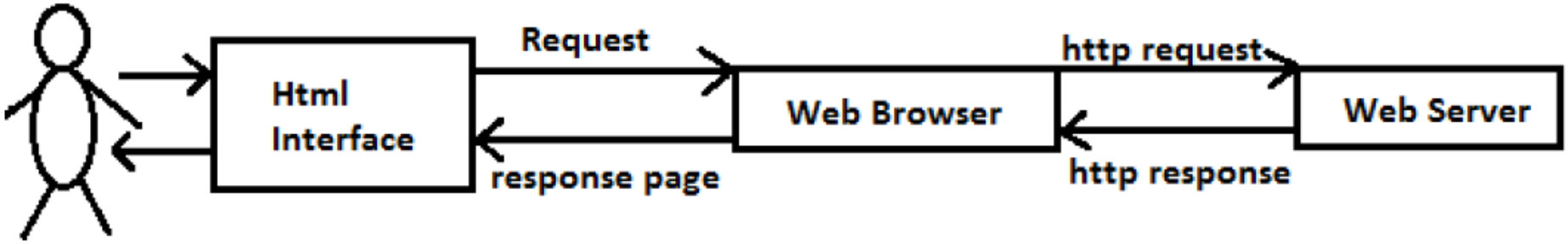

Рис. 14.3 - Схема работы веб-приложения

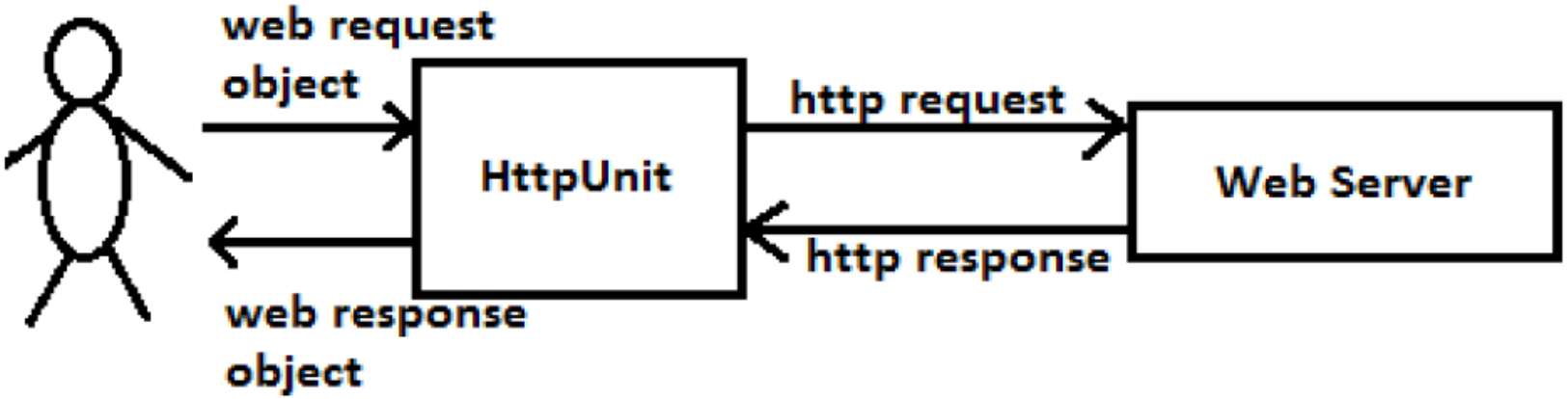

Рис. 14.4 - Схема работы инструмента httpUnit

Тестирование ПО

К преимуществам httpUnit можно отнести следующие:

– высокая скорость работы скриптов, а также их работа в фоновом режиме, поскольку отсутствует посредник в виде браузера;

– тестирование «чистого» приложения без учета особенностей браузера;

– бесплатность, поскольку httpUnit является свободно распространяемым инструментом автоматизации тестирования.

Тестирование ПО

Универсальных ср-ва автоматизации тестирования

Самым популярным инструментом для проведения различных видов автоматизированного тестирования является TestComplete. Автоматизация модульного тестирования является лишь небольшой частью его возможностей, которую можно реализовать двумя способами:

1) Подключение тестов, ранее написанных в других средах, например, в MSTest либо NUnit. Для запуска таких тестов существует два варианта:

– вариант 1 – указать в настройках TestComplete расположение этого стороннего теста (Tools\Options\ UnitTesting), создать проект для модульного тестирования, указав, что тесты были

Тестирование ПО

созданы в другом инструменте. Далее подключаем ранее созданные библиотеки dll, содержащие тесты. При таком варианте инструменту TestComplete отводится роль средства по обработке, сбору и представлению результатов в удобной форме;

– вариант 2 – в исходном тестовом инструменте подключить библиотеку AutomatedQA.TestComplete.UnitTesting.dll, указать, какие классы будут видимы для TestComplete.

Затем производится создание и настройка проекта в TestComplete и далее запуск тестов.

На панели справа можно выбрать запуск всех тестов либо только выбранных.

Тестирование ПО

При таком варианте TestComplete принимает непосредственное участие в запуске тестов и обработке результатов, причем процесс происходит несколько быстрее.

2) Непосредственное создание модульных тестов в TestComplete. Для этого необходимо реализовать следующие этапы:

а) Создание проекта. В TestComplete в окне Create New Project указывается имя, расположение и скриптовый язык будущего тестового проекта. В открывшемся окне Project Wizard можно указать, какие элементы проекта будут в него включаться (скрипты, список тестируемых приложений и т. д.).

Тестирование ПО

После нажатия кнопки Finish, TestComplete создаёт новый проект и отображает его содержимое в панели Project Explorer.

b) Подготовка модульных тестов. TestComplete может видеть внутренние объекты тестируемого приложения и позволяет использовать три варианта написания тестов:

– вариант 1 – создать метод, который тестирует остальные методы класса. Такой вариант реализуем, поскольку TestComplete может вызывать функции и методы непосредственно из кода приложения и обрабатывать исключения, при неправильной работе тестируемого метода. В итоге не придется писать дополнительные тестовые классы;

Тестирование ПО

– вариант 2 – создать функцию, которая будет тестировать методы класса и генерировать исключение при ошибке. Функция добавляется в тот же класс, где содержатся методы. Преимущества данного метода: количество классов в проекте не изменяется, функция имеет доступ к методам с модификаторами private и protected;

– вариант 3 – это создание отдельного класса для тестирования исходного класса. Преимущества: тестируемый и тестирующий классы разделены, что дает возможность легко деактивировать модули, которые содержат тесты.

Тестирование ПО

c) Подготовка тестируемого приложения. Для этого необходимо открыть тестируемый проект, в разделе Bin/Extentions добавить ссылку на AutomatedQA.TestComplete.UnitTesting.dll, установить свойство CopyLocal = True. Затем необходимо вызвать метод AddClasses(), принадлежащий объекту UnitTesting, например, в следующей реализации:

```
Using TestComplete;
Type[] types = {typeof(MyTestClass)};
UnitTesting.AddClasses(types);
```

После добавления классов все методы, принадлежащие им, будут видны для TestComplete.

Тестирование ПО

Затем необходимо скомпилировать приложение и запустить его.

d) Загрузка тестов. В созданный проект TestComplete необходимо добавить тестируемое приложение (Tested Applications\Add\New Item), которое должно быть запущено. Добавить объект TCUnitTest в раздел UnitTesting. В появившемся окне выбрать процесс, соответствующий тестируемому приложению. Загрузить доступные тесты, выбрав для этого Menu->Mode->Load. Если был выбран автоматический режим, то тесты будут загружены автоматически.

e) Запуск тестов. Для удобства запуска тестов можно добавить несколько строк кода в модуль Script, в результате чего TestComplete будет автоматически

Тестирование ПО

запускать тестируемое приложение и прогонять доступные тесты. Результаты каждого прогона модульных тестов сохраняются в UnitTesting логах и могут быть открыты для анализа.

Планирование функционального тестирования

После появления первой рабочей версии требований к программному продукту тестировщики приступают к подготовительным работам для проведения функционального тестирования, т. е. планируют свою тестовую деятельность. На этом этапе необходимо определиться со следующими моментами:

Тестирование ПО

– какой программный продукт будет тестироваться, каково его назначение;

– как будет использоваться продукт;

– какие части продукта будут тестироваться, а какие нет;

– какие методы и техники тестирования наиболее эффективны для данного проекта;

– какие риски могут возникнуть в процессе тестирования (риск – это возможная ситуация, которая негативно повлияет на результативность процесса либо приведет к увеличению сроков реализации, например, эпидемия гриппа в осенний период или период отпусков летом).

Тестирование ПО

Все указанные выше моменты оформляются в тестовый план и рассылаются ключевым участникам проекта. Параллельно тестировщиками разрабатываются тестовые случаи.

Тестовый план

Тестовый план – это документ, согласно которому будут проводиться все действия по тестированию. Ответственным за разработку тестового плана, как правило, является руководитель команды тестирования.

Можно привести примерное содержание тестового плана:

Тестирование ПО

1. Объемы тестирования (перечень тестируемых и нетестируемых компонент).
2. Критерии качества (какое количество ошибок может быть отклонено при сдаче продукта).
3. Риски тестирования.
4. Документация тестирования (тестовый план, тестовые случаи, отчеты об ошибках).
5. Стратегия тестирования (это самый большой и самый важный пункт, в котором описываются применяемые виды тестирования и градация тестов).
6. Ресурсы (перечень человеческих ресурсов с разделением прав и обязанностей, а также

Тестирование ПО

аппаратные ресурсы: сервера, рабочие станции, инструменты тестирования, описание тестового окружения).

7. График тестирования (основывается на графике выпуска версий).

Составленный тестовый план должен быть проверен и удовлетворять следующим критериям:

– является полным, корректным, недвусмысленным;

– цели каждого вида тестирования определены;

– четко определена стратегия тестирования;

– является реально выполнимым;

– определено тестовое оборудование;

Тестирование ПО

– оговорены условия прекращения тестирования;

– определены ресурсы (человеческие, аппаратные, программные) для тестирования.