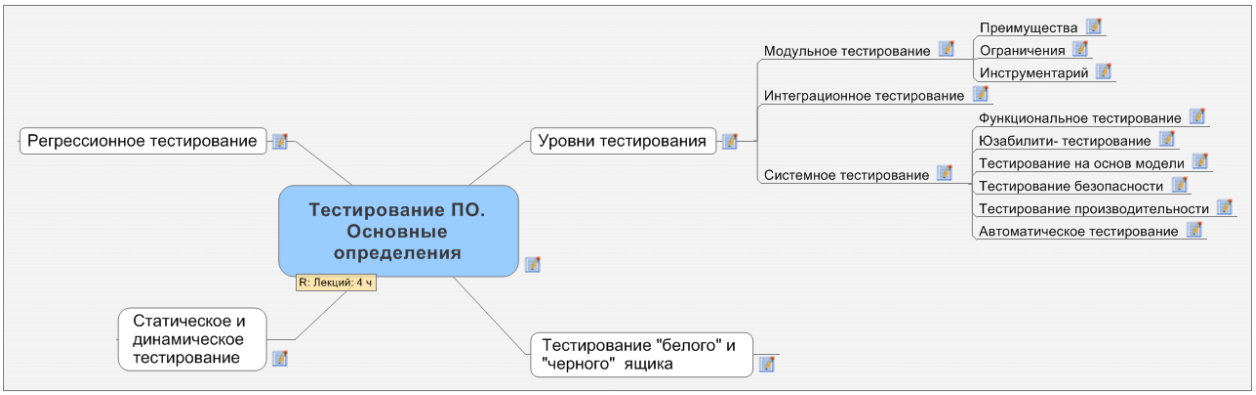
Лекция 10. Тестирование ПО. Основные определения



Resource: Лекций: 4 ч

Тестирование ПО- это процесс выполнения ПО в контролируемых условиях с целью получения ответа на вопрос "Ведет ли ПО себя так, как специфицировано?".

Термин ***ошибка (bug)*** часто используется в отношении проблем и сбоев в компьютере. Различают программные и аппаратные ошибки. При тестировании программного обеспечения речь идет о программных ошибках.

Международный стандарт ANSI/IEEE–729–83 разделяет все ошибки в разработке программ на следующие:

- ***Ошибка (error)*** – состояние программы, при котором выдается неправильные результаты, причиной которых являются изъяны ***(flaw)*** в операторах программы или в технологическом процессе ее разработки, что приводит к неправильной интерпретации исходной информации, а следовательно и к неверному решению.
- ***Дефект (fault)*** в программе является следствием ошибок разработчика на любом из этапов разработки и может содержаться в исходных или проектных спецификациях, текстах кодов программ, эксплуатационной документация и т.п. Дефект обнаруживается в процессе выполнения программы.
- ***Отказ (failure)***– это отклонение программы от функционирования или невозможность программы выполнять функции, определенные требованиями и ограничениями ирассматривается как событие, способствующее переходу программы в неработоспособное состояние из–за ошибок, скрытых в ней дефектов или сбоев в среде функционирования.

Тестовый случай (test-case)- пара ***(входные данные- ожидаемый результат)***, в которой входные данные- это описание данных, подаваемых на вход системы, а ожидаемый результат- описание выходных данных, которая система должна предъявить в ответ на соответсвтующий ввод.

Выполнение (прогон) тестового случая- это сеанс работы системы в рамках которого на вход системы подаются наборы данных, предусмотренные спецификацией тестового случая, и фиксируются результаты их обработки, которые затем сравниваются с ожидаемыми результатами, указанными в тестовом случае.

Если фактический результат отличается от ожидаемого, значит обнаружен отказ, т.е. тестируемая система не прошла испытания на заданном ткестовом случае. Если полученный результат соывпадает с ожидаемым, значит систма прошла испытание на заданном тестовом случае.

Из тестовых случаев формируются ***тествые наборы (test suits).*** Тестовые наборы организуются в определенном порядке, отражающем свойства тестовых случаев. Если система увпешно справилась со всеми тестовыми случаями из набора, то она успешно прошла испытания на тестовом наборе.

1 Уровни тестирования

Модульное тестирование (юнит-тестирование) — тестируется минимально возможный для тестирования компонент, например, отдельный класс или функция

Интеграционное тестирование — проверяет, есть ли какие- либо проблемы в интерфейсах и взаимодействии между интегрируемыми компонентами — например, не передается информация, передаётся некорректная информация.

Системное тестирование — тестируется интегрированная система на её соответствие исходным требованиям

- ***Альфа-тестирование*** — имитация реальной работы с системой штатными разработчиками, либо реальная работа с системой потенциальными пользователями/заказчиком на стороне разработчика. Часто альфа-тестирование применяется для законченного продукта в качестве внутреннего приёмочного тестирования. Иногда альфа-тестирование выполняется под отладчиком или с использованием окружения, которое помогает быстро выявлять найденные ошибки. Обнаруженные ошибки могут быть переданы тестировщикам для дополнительного исследования в окружении, подобном тому, в котором будет использоваться ПО.
- ***Бета-тестирование*** — в некоторых случаях выполняется распространение версии с ограничениями (по функциональности или времени работы) для некоторой группы лиц, с тем чтобы убедиться, что продукт содержит достаточно мало ошибок. Иногда бета- тестирование выполняется для того, чтобы получить обратную связь о продукте от его будущих пользователей.

1.1 Модульное тестирование

Модульное или юнит-тестирование (unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея 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 написанных и оттестированных местах программы, а также облегчает локализацию и устранение таких ошибок.

1.1.1 Преимущества

Цель юнит-тестирования — изолировать отдельные части программы и показать, что по отдельности эти части — работоспособны. Этот тип тестирования обычно выполняется программистами. Достоинствами юнит-тестирования является

- Поощрение изменений

Юнит-тестирование позволяе тпрограммистам проводить реорганизацию кода программы, будучи уверенными, что модуль по-прежнему работает корректно. Это поощряет программистов к изменениям кода, поскольку достаточно легко проверить, что код работает и после изменений.

- Упрощение интеграции

Юнит-тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх»: сначала тестируются отдельные части программы, затем программа в целом.

- Документирование кода

Юнит-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***.

В объектно-ориентированном программировании mock- объектом называют специальный объект, создававемый для организации тестирования программного модуля. Такой объект имитирует поведение реального объекта, с которым будет взаимодействовать тестируемый модуль.
Это приводит к менее связанному коду, минимизируя зависимости в системе.

1.1.2 Ограничения

Важно понимать, что юнит-тестирование не выловит *все* ошибки. По определению, оно тестирует только модули сами по себе. Тем самым, оно не выявит ошибки интеграции, проблемы производительности или любые другие проблемы системы в целом.

Кроме того, довольно трудно предугадать все варианты исходных данных, которые могут быть переданы модулю в реальной работе.

Юнит-тестирование будет эффективным только при использовании совместно с другими способами тестирования.

1.1.3 Инструментарий

Для большинства популярных языков программирования_высокого уровнясуществуют ин струменты и библиотеки юнит- тестирования. Некоторые из них:

- Для Java
  - JUnit JUnit.org
  - TestNGtestNG.org
  - JavaTESK UniTESK.ru
- NUnit [1]—для языков платформы.NET:C#, Visual Basic .NETи др.
- Для C
  - CTESK UniTESK.ru
- Для C++
  - CPPUnit[2]
  - Boost Test [3]
  - googletest[4]
  - Symbian[5]—фреймворк для Symbian OS всех версий.
- DUnit [6]—дляDelphi
- Для Perl
  - Test [7]
  - Test::Simple [8]
  - Test::Unit [9]

        - Test::Unit::Lite [10]
- Для PHP
    - SimpleTest [11]
    - PHPUnit [12]
- Для Python
    - PyUnit [13]
    - PyTest [14]
    - Nose [15]
- vbUnit [16]—Visual Basic
- utPLSQL [17]—PL/SQL
- Для ActionScript 2.0 —язык_сценариев, используемый_виртуальной_машиной_Adobe Flash Playerверсии 7 и 8
    - AsUnit [18]
    - AS2Unit [19]
- Для ActionScript 3.0 —скриптовый язык, используемый_виртуальной_машинойAdobe Flash Player_версии 9
    - FlexUnit [20]
    - AsUnit [21]
- Test::Unit [22] — для Ruby
- JsUnit [23]— для JavaScript

1.2 Интеграционное тестирование

Интеграционное тестирование (Integration testing, Integration and Testing, I&T) — одна из фаз тестирования программного обеспечения, при котором отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию.

Интеграционное тестирование в качестве входных данных использует модули, над которыми было проведено модульное тестирование, группирует их в более крупные множества, выполняет тесты, определенные в плане тестирования для этих множеств, и представляет их в качестве выходных данных и входных для последующего системного тестирования.

Целью интеграционного тестирования является проверка соответствия проектируемых единиц функциональным, приемным и требованиям надежности. Тестирование этих

проектируемых единиц — объединения, множества или группа модулей — выполняются через их интерфейс, используя тестирование «черного ящика».