**Unit Testing**

**Unit testing** предназначен для того, чтобы проверить, что отдельные части кода работают правильно. Целью unit testing-а является изоляция каждой из частей программы для того, чтобы убедиться, что отдельные её части являются правильными.

Блочное (модульное, unit) тестирование наиболее понятное для программиста. Фактически это тестирование методов какого-то класса программы в изоляции от остальной программы.

**Пример:**

**Task:** необходимо написать функцию, которая будет высчитывать значение скорости по формуле: *speed=distance/time.*

**Testing:** проверить, что при разных значениях *distance* и *time*, функция будет возвращать корректные значения.

double GetValue(double nValue1, double nValue2)

{

      return nValue1/nValue2;

}

double GetValue(double nValue1, double nValue2)

{

      return (nValue2 != 0) ? nValue1/nValue2 : 0;

}

**Defect:** возникает эксепшен при значении time=0.

**ISD part:** программисты пишут юнит тест кейсы и проходят их.

Essence:

* Applicable to modular design: unit tests inspects individual modules;
* Locate error in smaller region:
* In an integrated system in may be not easier to determine which module has caused fault;
* Reduces debugging efforts (ускоряет процесс отладки приложения и рефакторинга)

**Integration Testing**

**Integration testing -** это вид тестинга, где отдельные программные модули скомбинированы и тестируются, как цельная группа. Два или больше модулей или функций тестируются вместе для того, чтобы найти дефекты в интерфейсах, взаимодействиях, переходах между модулями или функциями.

**Пример:** скрипты базы данных, основной код приложения и GUI компоненты были разработаны разными программистами. Есть необходимость протестировать, как эти 3 компонента будут взаимодействовать между собой.

**Integration Testing Procedure:** Combine three parts into one system and verify interfaces (check interaction with database and GUI).

**Defect:** “Materials” button action returns not list of books but list of available courses.

ISD:

**System Testing**

Validating a fully developed system against its requirements.

System testing рассматривает систему, как единое целое с целью проверить работу системы на соответствие ее реквайрментам. Как правило, на входе system testing получает интегрированные ее компоненты, для которых успешно был пройден Integration testing, а также необходимым является то, что система успешно работает с любыми hardware-системами. Системный тестинг призван искать дефекты как между ее составляющими, а также системы в целом.

Системное тестирование — это тестирование программы в целом. Для небольших проектов это, как правило, ручное тестирование — запустил, пощелкал, убедился, что (не) работает. Можно автоматизировать.

Все программные продукты должны быть протестированы на каждом этапе разработки для того, чтобы убедиться, что система отвечает требованиям, что она надежна и безопасна для использования.

Software testing – это спланированная часть Software Development Life Cycle. Все тест планы и тест кейсы должны быть задокументированы и завершены, а тестирование проверено. Любые найденные дефекты должны быть исправлены в порядке их критичности. К System testing относят такие виды тестинга:

* Change Testing
* Regression Testing
* Exploratory Testing
* System Testing
* Automated Testing
* Suite Integration Testing

System testing производится предварительно для всех продуктов перед релизом для клиентов.

System testing может быть произведен как на нескольких модулях, так и на отдельных модулях.

System testing начинается, когда все критичные дефекты были исправлены и протестированы.

System testing начинается, когда финальный билд продукта был скомпилирован и проинсталлирован на тестовом environment-е, а также следующая документация была получена и подписана:

1. Тест план;
2. Scope документ;
3. Final Change List
4. Change testing summary report
5. Regression testing summary report
6. Testing analysis

**Change testing**

Change testing - тестирование изменения, вносимого программистом в программу, на соответствие заявленным спецификациям этого изменения. Проводится для тестирования:

* Дефектов
* Клиентских тасок

После проведения необходимых изменений, таких как исправление бага/дефекта, программное обеспечение должно быть пере тестировано для подтверждения того факта, что проблема была действительно решена. Такое тестирование необходимо проводить после установки программного обеспечения со сделанными изменениями, для подтверждения работоспособности приложения или правильности осуществленного исправления дефекта.

**Regression Testing**

**Regression testing** goal:

Assure that software functionality that previously worked as desired didn’t stop working;

It works in the same way that was previously planned.

Typically, regression bugs occur as an unintended consequence of program changes.

Does not belong to either unit test, integration test or system test. Instead, it is separate dimension to these three forms of testing. It is a running of test suite:

* After each change to the system or after each bug fix;
* Ensures that no new bug has been introduced during the change or the bug fix.

It assures:

* The new system performance is at least as good as the old system;
* Always used during phased system development.

Common methods of regression testing include re-running previously run tests and checking whether previously fixed faults have re-emerged.

**Task:** перепроверить какую-либо функциональность после фикса. Например, сохранение документа, после того как был исправлен дефект возможности сохранения с недопустимыми символами в названии файла.

**Regression Testing Procedure:** перепройти ТК на данную функциональность, а также другие кейсы, которые могут зависеть от данной функциональности.

**Defect:** Возможен любой тип дефекта. Например сохранить файл мы можем, а если уже есть документ с таким же именем, то программа перестала спрашивать пользователя о его желании заменить данный документ. Вместо этого она просто заменяет старый документ новым.

**Exploratory testing**

<http://www.slideshare.net/alikhan_nazeer80/exploratory-testing>

E. t. is useful for discovering new information.

Simultaneously:

* Learn about product;
* Learn the ways the product could fail
* Learn about the weakness of product
* Learn about how to test the product
* Test the product
* Report the defects
* **Develop new tests based on what you have learned so far**

It is not a technique. It’s way of thinking about testing.

In scripted testing tests are first designed and recorded. Then they may be executed at some later time or by different tester.

VS

In exploratory testing tests are designed and executed at the same time and they often are not recorded.

**Manual Testing - Automated testing**

Automated Testing – процесс проверки программного обеспечения на соответствие требованиям, который выполняется специальными программными средствами; тесты выполняются без мануального, ручного вмешательства.

Automated Test Case – реализация TC в скриптовом файле (или любом другом) в форме, которая может быть выполнена инструментами AT (например, программа SilkTest).

Manual Test Case (MTC) – выполнение тестового кейса в ручном виде.

Silktest - инструмент для автотестирования функциональности приложений.

Автоматическое тестирование в первую очередь предназначено для программ, для которых работоспособность и безопасность при любых входных данных являются наиважнейшими приоритетами: веб-сервер, клиент/сервер SSH, sandboxing, сетевые протоколы. (<http://habrahabr.ru/blogs/code_review/128503/>)

**Что может предложить автоматизация тестирования? Или выгоды автоматизации тестирования.**

Автоматизация тестирования может дать возможность выполнять определенные тестовые задачи намного быстрее и эффективнее чем это может быть сделано вручную. Это и так понятно, но, все-таки, существуют еще некоторые плюсы, которые люди не всегда осознают:

1. **Выполнение существующих регрессионных тестов.** Это должно быть самая очевидная задача автоматизации, в частности, когда программы исправляются и дополняются очень часто. Усилия, потраченные на выполнение существующих регрессионных тестов должны быть минимальны, для того, чтобы в нужный момент была возможность выбора  автоматизированных тестов и их запуск с минимальными ручными усилиями и в считанные минуты.
2. **Выполнение большого количества тестов более часто.** Известный факт, что часто-повторяемые задачи наиболее хорошо поддаются автоматизации. Четкая выгода в том, чтобы запускать большее количество тестов, в меньшие сроки, давая при этом косвенную возможность запускать тесты более часто.
3. **Выполнение тестов, которые будет сложно или невозможно сделать.**Предположим, что нам необходимо симулировать нагрузку в 500 пользователей на какую-то систему. Я думаю, что это будет невозможно или крайне затратно сделать ручным способом. С другой стороны, автоматические тесты могут выполнить все это в любой момент, и запустить их сможет любой человек, который даже не будет особо разбираться в логике системы. Другой пример невыполнимой работы может быть следующим. Каждый пользователь, смотря на графический интерфейс, интуитивно ожидает появление какой-то визуализации данных при определенный действиях, но пользователь (тестеровщик) может даже и не представлять, что за этим кроются какие-то скрытые механизмы, такие как, например, вызов и обработка событий, которые нельзя проверить визуально. События и их поведение просто невозможно проверить без использования специальных программных средств.
4. **Использовать сэкономленные ресурсы на более важные и интересные задачи.** Автоматизация обычных и скучных задач, например, повторяемых с разными вариациями данных, дает возможность полагаться на большую степень аккуратности при проведения тестирования и при этом улучшается моральное состояние людей, которые работаю с такими задачами. Также высококвалифицированные работники могут свободно вкладывать свои усилия в проектирование более сложных автоматических тестов, которые впоследствии будут выполняться. Ручная работа некуда не денется. Ручное тестирование является неотъемлемой частью автоматизации тестирования и наоборот. Всегда будут задачи, которые лучше выполнять вручную, более того ручное тестирование будет выполняться намного лучше, если автоматические тесты будут давать уверенность в том, что в протестированном участке нет дефектов.
5. **Постоянство и повторяемость тестов.** Автоматизация дает нам возможность постоянно выполнять тесты с четко одними и теми же данными (входные данные всегда будут те же, а вот выходные могут отличаться на протяжении некоторого времени разработки), чего нам (увы) не может гарантировать человек, выполняя эту же работу вручную, потому уверенность в тестировании повышается. Эти же тесты могут быть выполнены на разных операционных системах и использованием разных баз данных, что тоже даст уверенность в том, что кросс-платформенность продукта на высоком уровне.
6. **Повторное использование тестов.** Усилия, вложенные на принятие решений, что необходимо тестировать, проектирования и формирования тестов может дать возможность использовать или повторно использовать разработанные тесты. В свою очередь это сэкономит время разработки новых тестов. Что необходимо учитывать, что бы такие тесты были стабильными, иначе потраченное время на поддержку тестов будет большим.
7. **Быстрый вывод продукта на рынок.**Как только тесты будут автоматизированы, у вас появиться возможность повторного запуска и следственно общее время, потраченное на выполнение тестирования, будет меньшим.
8. **Повышающаяся уверенность.** Конечно же, осознание и понимание, что исчерпывающее количество автоматических тестов дают постоянно положительные результаты увеличивает доверие и уверенность в том, что не будет никаких непредсказуемых сюрпризов, когда программное обеспечение будет выпущено (конечно, подразумевается, что тесты выполняют свою функции и проверяют ПО на наличие ошибок).

**Functional Testing**

**Functional testing** его целью является проверка бизнес логики, необходимых алгоритмов, use cases в соответствии с реквайрментами.

Пример: протестировать Save в приложении Notepad

**Functional Testing Procedure:** test different flows of Save functionality (Save new file, save updated file, test Save As, save to protected folder, save with incorrect name, re-write existed document, cancel saving, etc.)

**Defect:** While trying to save file with not allowed name (use reserved symbols <, >, :, \, / in file name) no message is shown but Save dialog is closed. File is not saved.

Message “Invalid file name” should be shown and Save dialog should remain opened until correct file name is entered of save process is cancelled.

**Performance Testing**

Иногда для Performance тестинга используется тестинг hardware и software одновременно.

**Performance testing** применяется чтобы определить, насколько быстро некоторые части системы работают при некоторой частичной загрузке.

Целей у Performance тестинга может быть несколько. Во-первых, для того, чтобы продемонстрировать, что производительность системы соответствует требованиям. Проверить две системы, какая из них работает быстрее. Или найти, какие части системы являются причиной плохой производительности.

**Criteria:** сервер должен отвечать не более 2-х секунд, когда 100 пользователей одновременно ее используют. Сервер должен отвечать не более 5-х секунд, когда 300 пользователей одновременно ее используют

**Performance Testing Procedure:** имитироватьодновременное использование пользователями системы от 0 до 300, измеряя время отклика для 10, 50, 100, 240, 290 пользователей.

**Defect:** начиная с 200 время отклика системы составляет 10-15 сек

**Load testing**

**Load testing**, как правило, относится к моделированию использования ПО, имитируя доступ нескольким пользователями к программе одновременно. Подвергает систему достаточно репрезентативной загрузкой.

**Criteria:** Сервер должен позволить до 500 одновременных соединений.

**Load Testing Procedure:** эмулировать различное количество запросов к серверу, чтобы измерить время для 400, 450, 500-одновременно работающих пользователей.

**Defect:** Server возвращает “Request Time Out” начиная с 490 пользователей.

**Stress (**endurance**) testing**

**Stress testing:**

* Вводятся нестандартные значения, чтобы измерить возможности ПО;
* Объемы исходных данных, скорость ввода данных, время обработки, использования памяти и т.д. проверяются за пределами возможностей архитектуры.

**Stress testing -** этот вид тестинга используется, чтобы определить стабильность системы. Она включает в себя тестирование за пределами обычных оперативных возможностей для наблюдения результатов.

Систему подвергают нагрузкам, не смотря на ее ресурсы (объемы памяти, дисковых пространств, прерываний) необходимых для процесса загрузки. Идея заключается в стресс системы до предела, чтобы найти ошибки, которые могут привести потенциально вредным последствиям.

**Criteria:** Сервер должен позволить до 500 одновременных соединений.

**Stress Testing Procedure:** emulate amount of requests to server greater than pick value, for instance, check system behavior for 500, 510, and 550 concurrent users.

**Defect:** Server crashes starting from 500 concurrent requests and user’s data is lost. Data should not be lost even in stress situations. If possible, system crash also should be avoided.

**Usability Testing**

**Usability testing** является средством для измерения того, насколько хорошо и легко люди могут использовать систему (например, веб-страницы, компьютерный интерфейс, документ или устройства) по их прямому назначению, то есть Usability testing – тестирование меры удобства использования системы.

Юзабилити-тестирование обычно включает в себя измерение того, насколько хорошо испытуемые используют систему в четырех областях: время, точность, запоминаемость и эмоциональное состояние.

* *Time on Task* – Сколько времени людям потребуется, чтобы выполнить какое-либо действие? (напр., что-то купить, создать новый аккаунт и получить что-то)
* *Accuracy* – Сколько ошибок люби сделали? (и были ли они фатальными или имели бы возможность для восстановления информации)
* *Recall* – сколько информации пользователь помнит, после периодов неиспользования программы
* *Emotional Response* – Как пользователь себя чувствует после использования? (Уверенно? В стрессе? Порекомендует ли он эту систему другим пользователям?)

**Task:** ввести функциональность, которая позволяла бы просматривать цену каждой книги в списке для какого-либо курса.

**Implementation: кнопка** «Материалы» открывает диалог “Required Material”, который содержит список книг, предназначенных для определенного курса (имя книги и автор). Кнопка «Price» из “Required Material” окна диалога открывает еще одно диалоговое окно с ценой на выбранную книгу.

**Usability Note:** Материалы обычно просматривают чтобы сразу оценить, сколько книг необходимо, чтобы выучить курс и сколько они стоят. Было бы удобнее, если бы цена показывалась сразу в диалоге “Required Materials” возле каждой книги в списке, чтобы избежать выполнения операции открытия диалога еще раз.

**GUI Testing**

**GUI testing** проводится для проверки соответствия GUI приложения определенным стандартам. Общие стандарты для некоторых типов приложений (Web, Desktop) или специальным требованиям клиента могут быть основаниями для тестинга.

**Boundary Testing**

Граничное тестирование применяется для проверки поведения продукта на крайних значениях входных данных, таких как, например, максимумы и минимумы значений. Набор тестов для стресс-тестирования может включать граничные тесты.

Граничное тестирование может также включать тесты, проверяющие поведение системы на входных данных, выходящих за допустимый диапазон значений. При этом система должна определенным (заранее оговоренным) способом обрабатывать такие ситуации. Например, с помощью исключительной ситуации или сообщения об ошибке.

**Security testing**

**Тестирование безопасности** - это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

Общая стратегия безопасности основывается на трех основных принципах:

1. конфиденциальность
2. целостность
3. доступность

### Конфиденциальность

Конфиденциальность - это сокрытие определенных ресурсов или информации. Под конфиденциальностью можно понимать ограничение доступа к ресурсу некоторой категории пользователей, или другими словами, при каких условиях пользователь авторизован получить доступ к данному ресурсу.

### Целостность

Существует два основных критерия при определении понятия целостности:

1. **Доверие**. Ожидается, что ресурс будет изменен только соответствующим способом определенной группой пользователей.
2. **Повреждение и восстановление**. В случае когда данные повреждаются или неправильно меняются авторизованным или не авторизованным пользователем, вы должны определить на сколько важной является процедура восстановления данных.

### Доступность

Доступность представляет собой требования о том, что ресурсы должны быть доступны авторизованному пользователю, внутреннему объекту или устройству. Как правило, чем более критичен ресурс тем выше уровень доступности должен быть.

**Localization Testing**

Языки. Адаптирование приложения к новому месту и системным языковым настройкам.

**Alpha testing**

Проводится тестинг тимой и разработчиками внутри компании. Используется, как форма внутреннего приемочного тестирования.

**Smoke Testing**

**Smoke testing** проводится для принятия решения о дальнейшем тестировании. Это общий просмотр функциональности системы, чтобы принять решение об отклонениях в данной сборке. Его также называют Build Verification Test.

Тестеры просматривают основную функциональность системы, не вдаваясь в подробности. Задаются вопросами типа: «Возможно ли вообще протестировать данную функциональность?» «Открывается ли данное окно?» «Работает ли приложение, если нажимать на данные кнопки?» Нет необходимости проверять бизнес-логику, если на такие вопросы вы получите ответ «нет». Тогда приложение не рабочее и дальше продолжать тестирование нет смысла.

**Пример:** протестировать приложение блокнот.

**Smoke Testing Procedure:** нужно быстро проверить основные возможности приложения блокнот (run application, type text, open file, save file, edit file).

**Defect:** There is no ability to save file. Button “Save” does nothing.

**Acceptance testing**

**Acceptance testing** проводится конечными пользователями, заказчиком или клиентами с целью принять или не принять продукт. Этот вид тестинга может проводиться между двумя фазами разработки.

**Acceptance Criteria:**

* Продукт должен отвечать всем требованиям
* Интерфейс должен отвечать требованиям, написанным к требованиям GUI интерфейса.
* Приложение должно за адекватное время работать на hardware backend.

Приложение не должно содержать незавершенной функциональности, все должно работать в соответствии с требованиями.

**Defect:** возможны любые варианты дефектов.

**White Box, Black Box, Grey Box testing**

**ЧЕРНЫЙ ЯЩИК (black box)**

1. Тестировщик не знает, как устроено приложение.

2. Идеи для тестирования идут от предполагаемых паттернов (pattern — образец) поведения пользователей. Поэтому подход "Черный ящик" также называют поведенческим.

При подходе "Черный ящик" тестировщик не основывает идеи для тестирования на знании об устройстве и логике тестируемой части бэк-энда. Идеи формируются путем предположений о сценариях, которые будут реализовываться и применяться пользователями. Такие сценарии называются паттернами поведения пользователей.

**БЕЛЫЙ ЯЩИК (white box)**

также известен под именами Стеклянный ящик (glass/clear box), Открытый ящик (open box) и даже Никакой ящик (по box).

В отличие от "Черного ящика", при подходе "Белый ящик" тестировщик основывает идеи для тестирования на знании об устройстве и логике тестируемой части бэк-энда.

Таким образом, при белоящичном тестировании сценарии создаются с мыслью о том, чтобы протестировать определенную часть бэк-энда, а не определенный паттерн поведения пользователя.

В реальной жизни белоящичное тестирование проводится либо самими программистами, написавшими код, либо их коллегами с программистской квалификацией того же уровня. Кстати, юнит-тестирование, о котором мы говорили, — это часть white box-тестирования.

**СЕРЫЙ ЯЩИК (gray/grey box)**

Это подход, сочетающий элементы двух предыдущих подходов, это

• с одной стороны, тестирование, ориентированное на пользователя, а значит, мы используем паттерны поведения пользователя, т.е. применяем методику "Черного ящика";

• с другой — информированное тестирование, т.е. мы знаем, как устроена хотя бы часть тестируемого бэк-энда, и активно используем это знание.

Ярчайший пример

Допустим, мы тестируем функциональность "регистрация":

• заполняем все поля (имя, адрес, е-мейл и т.д.) и

• нажимаем кнопку "Зарегистрироваться".

Следующая страница — подтверждение, мол, дорогой Иван Иваныч, поздравляем, вы зарегистрированы.

Теперь вопрос: если мы видим страницу с подтверждением регистрации, то значит ли это, что регистрация была успешной? Ответ: нет, так как процесс регистрации с точки зрения нашей системы включает не только подтверждение на веб-странице, но и создание записи в базе данных, т.е. вывод, который стоит проверить, состоит из

• страницы с подтверждением и

• новой записи в базе данных.

Откуда мы почерпнем знание о логике создания записей в базе данных при регистрации? Например, из технической документации (документ о дизайне/архитектуре системы, документ о дизайне кода), общения с программистом, самого кода.

Как видно из последнего примера, подход "Серый ящик" — это дело хорошее, жизненное и эффективное. Деятельность большинства профессиональных тестировщиков интернет-проектов протекает именно в разрезе сероящичного тестирования.

**Alpha testing**

System testing is carried out by the test team within the developing organization.

**Beta testing**

**Beta testing** проводится после альфа тестирования. Версии приложения, называемые бета-версиями, представляются узкому кругу пользователей. Иногда бета-версии доступны любым пользователям чтобы получить лучший feedback от максимального количества разных пользователей.

System testing is performed by a select group of friendly customers.

Динамический vs Статический анализ системы

В качестве более эффективной альтернативы динамическому анализу часто рассматривают статический анализ. В случае статического анализа поиск возможных ошибок осуществляется без запуска исследуемой программы, например, по исходному коду приложения. Обычно при этом строится абстрактная модель программы, которая, собственно, и является объектом анализа. При этом следует подчеркнуть следующие характерные особенности статического анализа:

* Возможен раздельный анализ отдельно взятых фрагментов программы (обычно отдельных функций или процедур), что дает достаточно эффективный способ борьбы с нелинейным ростом сложности анализа.
* Возможны ложные срабатывания, обусловленные либо тем, что при построении абстрактной модели некоторые детали игнорируются, либо тем, что анализ модели не является исчерпывающим.
* При обнаружении дефекта возникают, во-первых, проблема проверки истинности обнаруженного дефекта ([false positive](http://ru.wikipedia.org/wiki/False_positive)) и, во-вторых, проблема воспроизведения найденного дефекта при запуске программы на определенных входных данных.

В отличие от статического анализа, динамический анализ осуществляется во время работы программы. При этом: 

* Для запуска программы требуются некоторые входные данные.
* Динамический анализ обнаруживает дефекты только на трассе, определяемой конкретными входными данными; дефекты, находящиеся в других частях программы, не будут обнаружены.
* В большинстве реализаций появление ложных срабатываний исключено, так как обнаружение ошибки происходит в момент ее возникновения в программе; таким образом, обнаруженная ошибка является не предсказанием, сделанным на основе анализа модели программы, а констатацией факта ее возникновения.

(<http://habrahabr.ru/blogs/code_review/128503/>)

### Fuzz testing (фаззинг)

**Фаззинг** – методика тестирования, при которой на вход программы подаются невалидные, непредусмотренные или случайные данные.  
  
Основная идея такого подхода состоит в том, чтобы случайным образом “мутировать”, т.е. изменять ожидаемые программой входные данные в произвольных местах. Все фаззеры работают примерно одинаковым образом, позволяя задавать некоторые ограничения на мутирование входных данных определенными байтами или последовательностью байтов. В качестве примера можно упомянуть:[zzuf (linux)](http://linux.die.net/man/1/zzuf), minifuzz (Windows), filefuzz (Windows). Фаззеры протоколов: [PROTOS(WAP, HTTP-reply, LDAP, SNMP, SIP) (Java)](https://www.ee.oulu.fi/research/ouspg/Protos), [SPIKE (linux)](http://www.blackhat.com/presentations/bh-usa-02/bh-us-02-aitel-spike.ppt). Фреймворки фаззеров: [Sulley (фреймворк для создания сложных структур данных)](http://code.google.com/p/sulley/).  
**Fuzz testing** – предтеча автоматического тестирования, метод “грубой силы”. Из преимуществ данного подхода выделяется только его простота. Очевидным же недостатком является то, что фаззер ничего не знает о внутреннем устройстве программы и, в конечном итоге, для полного покрытия кода вынужден перебирать *астрономическое количество* вариантов (как нетрудно догадаться, полный перебор растет экспоненциально от размера входных данных O(c^n),c>1). (<http://habrahabr.ru/blogs/code_review/128503/>)