Тестирование программного обеспечения

ФИО преподавателя: Аксёнов Александр Вячеславович

e-mail: aksenov@mirea.ru

Основные понятия

Тестирование программного обеспечения (Software Testing) - проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.

В более широком смысле, **тестирование** - это одна из техник контроля качества, включающая в себя активности по:

- планированию работ (**Test Management**),
- проектированию тестов (**Test Design**),
- выполнению тестирования (**Test Execution**) и
- анализу полученных результатов (**Test Analysis**).

Классификация видов тестирования:

1. По знанию внутренностей системы:

- черный ящик *(black box testing);*
- серый ящик *(grey box testing);*
- белый ящик *(white box testing).*

2. По объекту тестирования:

- функциональное тестирование *(functional testing);*
- тестирование интерфейса пользователя *(UI testing);*
- тестирование локализации *(localization testing);*
- тестирование скорости и надежности *(load/stress/performance testing);*
- тестирование безопасности *(security testing);*
- тестирование опыта пользователя *(usability testing);*
- тестирование совместимости *(compatibility testing).*

Классификация видов тестирования:

3. По субъекту тестирования:

• альфа-тестировщик *(alpha tester);*

• бета-тестировщик *(beta tester).*

4. **По времени проведения тестирования:**

• **до** передачи пользователю — альфа-тестирование *(alphatesting);*

- тест приемки *(smoke test, sanity test или confidence test);*

- тестирование новых функциональностей *(new feature testing);*

- регрессивное тестирование *(regression testing);*

- тест сдачи *(acceptance or certification test);*

• **после** передачи пользователю — бета-тестирование *(beta testing).*

5. По критерию "позитивности" сценариев:

• позитивное тестирование *(positive testing);*

• негативное тестирование *(negative testing).*

6. По степени изолированности тестируемых компонентов:

- компонентное тестирование *(component testing);*
- интеграционное тестирование *(integration testing);*
- системное (или энд-ту-энд) тестирование *(system or endto-end testing).*

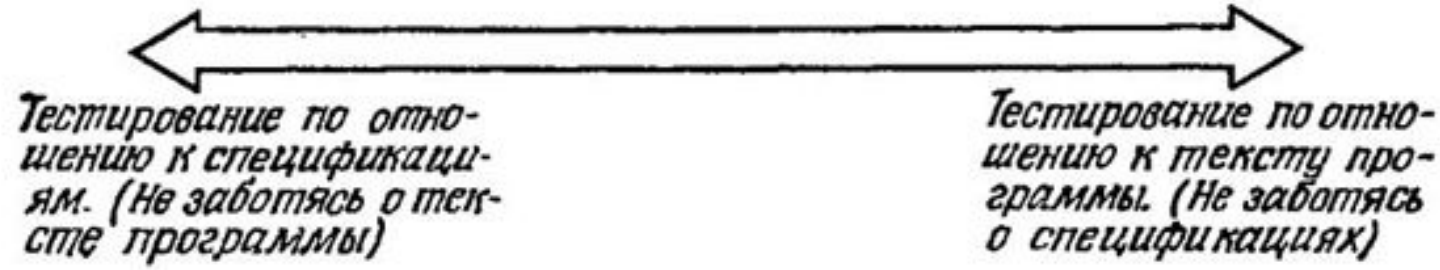
7. По степени автоматизированности тестирования:

- ручное тестирование *(manual testing);*
- автоматизированное тестирование *(automated testing);*
- смешанное/полуавтоматизированное тестирование *(semi automated testing).*

8. По степени подготовки к тестированию:

- тестирование по документации *(formal/documented testing);*
- эд хок-тестирование (интуитивное) *(ad hoc testing).*

Подходы к проектированию тестов



Левый крайний подход заключается в том, что тесты проектируются только на основании изучения спецификаций ПС: спецификация анализа ПС, описания архитектуры и спецификаций модулей. Содержимое исходных текстов модулей программы при этом никак не учитывается.

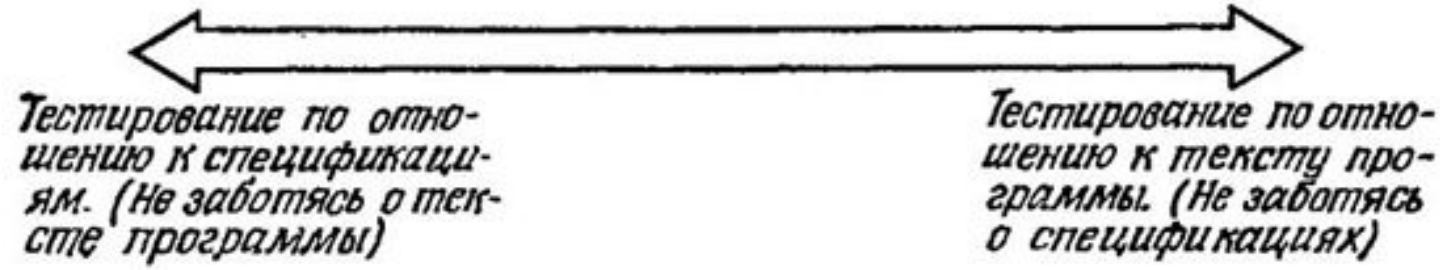
Такой подход получил название *тестирование* ***«черного ящика»***.

Фактически такой подход требует полного перебора всех наборов входных данных. На практике этот метод не применим.

Например, для исчерпывающего тестирования программы с 10 входными величинами, каждая из которых принимает по 10 значений, потребовалось бы 10^{10} тестовых наборов.

Пропуск части тестов чреват тем, что некоторые участки тестируемой программы не будут работать, и, следовательно, содержащиеся в них ошибки не будут проявляться.

Подходы к проектированию тестов



Правый крайний подход заключается в том, что тесты проектируются на основании изучения исходных текстов программ с целью протестировать все пути выполнения каждой программы входящей в состав ПС.

Такой подход получил **название *тестирование "белого ящика"***. Такой подход также не применим на практике, так как число различных путей выполнения программы может оказаться чрезвычайно много.

В качестве примера рассмотрим цикл в программе, который выполняется k раз, в теле которого содержится n ветвлений.

Число путей выполнения программы можно вычислить по следующей формуле:

$$M = k^{2^n}$$

Оптимальный подход к проектированию тестов

Оптимальный подход к проектированию тестов расположен между этими крайними подходами, но ближе к левому краю. При этом подходе значительная часть тестов проектируется по спецификациям. Оставшаяся часть тестов разрабатывается по исходным текстам программ.

При этом в первом случае этот подход базируется на следующих принципах:

- на каждую подпрограмму из функциональной спецификации ПС – хотя бы один тест;
- на каждую область изменения какой-либо входной величины и на их границы – хотя бы один тест;
- на каждую исключительную ситуацию из функциональной спецификации ПС – хотя бы один тест.

Во втором случае эта стратегия базируется на принципе: каждый оператор каждой программы ПС должен быть выполнен хотя бы в одном тесте.

Серый ящик

Тестировочное покрытие *(test coverage) состоит из двух вещей:*

- а. Покрытие возможных сценариев (черный ящик).
- б. Покрытие исполнения тест-кейсов (белый ящик).

Покрытие возможных сценариев — это в большинстве случаев абстрактная величина, так как в большинстве же случаев невозможно даже подсчитать, сколько понадобится тест-кейсов, чтобы обеспечить 100%-ю проверку.

Покрытие возможных сценариев может увеличиться либо уменьшиться путем прибавления либо отнятия уникального тест-кейса:

- который тестирует реальный сценарий использования ПО и
- который не является дубликатом другого тест-кейса.

Покрытие исполнения тест-кейсов — это **всегда величина конкретная,** и выражается она процентным отношением исполненных тест-кейсов к общему количеству тест-кейсов.

Серый ящик

Симбиоз использования подходов "Черный ящик" и "Белый ящик" увеличивает **покрытие возможных сценариев**

- **количественно,** потому что появляется большее количество тест-кейсов;
- **качественно,** потому что ПО тестируется принципиально разными подходами: с точки зрения пользователя ("Черный ящик") и с точки зрения кода ("Белый ящик").

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 кода, и активно **используем** это знание.

Функциональные виды тестирования

Функциональные тесты базируются на функциях и особенностях, а также взаимодействии с другими системами, и могут быть представлены на всех уровнях тестирования.

Функциональные виды тестирования рассматривают внешнее поведение системы.

1. Функциональное тестирование (Functional Testing)

Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом.

Функциональные тесты основываются на функциях, выполняемых системой, и могут проводиться на всех уровнях тестирования (*компонентном, интеграционном, системном, приемочном*). Как правило, эти функции описываются в требованиях, функциональных спецификациях или в виде вариантов использования системы (**use cases**).

Тестирование функциональности может проводится в двух аспектах:

- требования
- бизнес-процессы

1. Функциональное тестирование (Functional Testing)

Преимущества функционального тестирования:

- имитирует фактическое использование системы;

Недостатки функционального тестирования:

- возможность упущения логических ошибок в программном обеспечении;
- вероятность избыточного тестирования.

Достаточно распространенной является автоматизация функционального тестирования.

2. Тестирование безопасности (Security and Access Control 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.

Принципы безопасности программного обеспечения

Общая стратегия безопасности основывается на трех основных принципах:

- конфиденциальность
- целостность
- доступность

Принципы безопасности ПО

- ***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.

Виды уязвимостей

- **XSS (Cross-Site Scripting)** - это вид уязвимости программного обеспечения (Web приложений), при которой, на генерированной сервером странице, выполняются вредоносные скрипты, с целью атаки клиента.
- **XSRF / CSRF (Request Forgery)** - это вид уязвимости, позволяющий использоваться недостатки HTTP протокола, при этом злоумышленники работают по следующей схеме: ссылка на вредоносный сайт устанавливается на странице, пользующейся доверием у пользователя, при переходе по вредоносной ссылке выполняется скрипт, сохраняющий личные данные пользователя (пароли, платежные данные и т.д.), либо отправляющий СПАМ сообщения от лица пользователя, либо изменяет доступ к учетной записи пользователя, для получения полного контроля над ней.
- **Code injections (SQL, PHT, ASP и т.д.)** - это вид уязвимости, при котором становится возможно осуществить запуск исполняемого кода с целью получения доступа к системным ресурсам, несанкционированного доступа к данным либо выведения системы из строя.
- **Server-Side Includes (SSI) Injection** - это вид уязвимости, использующий вставку серверных команд в HTML код или запуск их напрямую с сервера.
- **Authorization Bypass** - это вид уязвимости, при котором возможно получить несанкционированный доступ к учетной записи или документам другого пользователя.

3. Тестирование взаимодействия (Interoperability Testing)

Тестирование взаимодействия – это функциональное тестирование, проверяющее способность приложения взаимодействовать с одним и более компонентами или системами и включающее в себя тестирование совместимости (compatibility testing) и интеграционное тестирование (integration testing).

Программное обеспечение с хорошими характеристиками взаимодействия может быть легко интегрировано с другими системами, не требуя каких-либо серьезных модификаций. В этом случае, количество изменений и время, требуемое на их выполнение, могут быть использованы для измерения возможности взаимодействия.

Нефункциональные виды тестирования

- **Нагрузочное тестирование**
- **Тестирование Установки или Installation Testing**
- **Тестирование удобства пользования или Usability Testing**
- **Тестирование на отказ и восстановление или Failover and Recovery Testing**
- **Конфигурационное тестирование или Configuration Testing**

Нагрузочное тестирование

Нагрузочное тестирование или **тестирование производительности** - это автоматизированное тестирование, имитирующее работу определенного количества бизнес пользователей на каком либо общем (разделяемом ими) ресурсе.

Виды нагрузочного тестирования:

- **Тестирование производительности** (**Performance testing**)
- **Стрессовое тестирование** (**Stress Testing**)
- **Объемное тестирование** (**Volume Testing**)
- **Тестирование стабильности или надежности** (**Stability / Reliability Testing**)

Тестирование производительности

Задачей тестирования производительности является определение масштабируемости приложения под нагрузкой, при этом происходит:

- измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций
- определение количества пользователей, одновременно работающих с приложением
- определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций)
- исследование производительности на высоких, предельных, стрессовых нагрузках

Стрессовое тестирование

Стрессовое тестирование позволяет проверить насколько приложение и система в целом работоспособны в условиях стресса и также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса.

Стрессом в данном контексте может быть повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера.

Также одной из задач при стрессовом тестировании может быть оценка деградации производительности, таким образом цели стрессового тестирования могут пересекаться с целями тестирования производительности.

Объемное тестирование

Задачей объемного тестирования является получение оценки производительности при увеличении объемов данных в базе данных приложения, при этом происходит:

- измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций
- может производиться определение количества пользователей, одновременно работающих с приложением

Тестирование стабильности

Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки.

Времена выполнения операций могут играть в данном виде тестирования второстепенную роль.

При этом на первое место выходит отсутствие утечек памяти, перезапусков серверов под нагрузкой и другие аспекты влияющие именно на стабильность работы.

Тестирование Установки

Тестирование установки направленно на проверку успешной инсталляции и настройки, а также обновления или удаления программного обеспечения.

1. Проверка инструкции для инсталляции.
2. В распределенных системах требуется план установки с предусмотренным откатом. Его тоже нужно тестировать.

Тестирование установки можно назвать одной из важнейших задач по обеспечению качества программного обеспечения.

Особенности инсталляторов

Инсталлятор - это "обычная" программа, основные функции которой - Установка (Инсталляция), Обновление и Удаление (Деинсталляция) программного обеспечения.

Являясь обычной программой, инсталлятор обладает рядом особенностей, среди которых стоит отметить следующие:

- Глубокое взаимодействие с операционной системой и зависимость от неё (файловая система, реестр, сервисы и библиотеки)
- Совместимость как родных, так и сторонних библиотек, компонент или драйверов, с разными платформами
- Удобство использования: интуитивно понятный интерфейс, навигация, сообщения и подсказки
- Дизайн и стиль инсталляционного приложения
- Совместимость пользовательских настроек и документов в разных версиях приложения

Особенности инсталляторов

Список рисков, который покажет всю значимость корректной работы инсталляторов:

- риск потери пользовательских данных
- риск вывода операционной системы из строя
- риск неработоспособности приложения
- риск не корректной работы приложения

Если хочется подробностей про тестирование инсталляторов - http://www.protesting.ru

Тестирование удобства пользования

Тестирование удобства пользования - это метод тестирования, направленный на установление степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий.

Уровни проведения

- Проверка удобства использования может проводиться как по отношению к готовому продукту, посредством тестирования черного ящика,
- так и к интерфейсам приложения (API), используемым при разработке - тестирование белого ящика.

Тестирование удобства пользования

Тестирование удобства пользования дает оценку уровня удобства использования приложения по следующим пунктам:

- **производительность, эффективность** (**efficiency**) - сколько времени и шагов понадобится пользователю для завершения основных задач приложения, например, размещение новости, регистрации, покупка и т.д.? (*меньше - лучше*)
- **правильность** (**accuracy**) - сколько ошибок сделал пользователь во время работы с приложением? (*меньше - лучше*)
- **активизация в памяти** (**recall**) – как много пользователь помнит о работе приложения после приостановки работы с ним на длительный период времени? (*повторное выполнение операций после перерыва должно проходить быстрее чем у нового пользователя*)
- **эмоциональная реакция** (**emotional response**) – как пользователь себя чувствует после завершения задачи - растерян, испытал стресс? Порекомендует ли пользователь систему своим друзьям? (*положительная реакция - лучше*)

Тестирование на отказ и восстановление

проверяет тестируемый продукт с точки зрения способности противостоять и успешно восстанавливаться после возможных сбоев, возникших в связи с ошибками программного обеспечения, отказами оборудования или проблемами связи (например, отказ сети).

Целью данного **вида тестирования** является проверка систем восстановления (или дублирующих основной функционал систем), которые, в случае возникновения сбоев, обеспечат сохранность и целостность данных тестируемого продукта.

Тестирование на отказ и восстановление очень важно для систем, работающих по принципу “24x7”.

Методика подобного **тестирования** заключается в симулировании различных условий сбоя и последующем изучении и оценке реакции защитных систем.

Тестирование на отказ и восстановление

Объектом тестирования в большинстве случаев являются весьма вероятные эксплуатационные проблемы, такие как:

- Отказ электричества на компьютере-сервере
- Отказ электричества на компьютере-клиенте
- Незавершенные циклы обработки данных (прерывание работы фильтров данных, прерывание синхронизации).
- Объявление или внесение в массивы данных невозможных или ошибочных элементов.
- Отказ носителей данных.

Тестирование на отказ и восстановление

Технически реализовать тесты можно следующими путями:

- Симулировать внезапный отказ электричества на компьютере (обесточить компьютер).
- Симулировать потерю связи с сетью (выключить сетевой кабель, обесточить сетевое устройство)
- Симулировать отказ носителей (обесточить внешний носитель данных)
- Симулировать ситуацию наличия в системе неверных данных (специальный тестовый набор или база данных).

Во всех вышеперечисленных случаях, по завершении процедур восстановления, должно быть достигнуто определенное требуемое состояние данных продукта:

- Потеря или порча данных в допустимых пределах.
- Отчет или система отчетов с указанием процессов или транзакций, которые не были завершены в результате сбоя.

Конфигурационное тестирование

Конфигурационное тестирование— специальный вид тестирования, направленный на проверку работы программного обеспечения при различных конфигурациях системы (заявленных платформах, поддерживаемых драйверах, при различных конфигурациях компьютеров и т.д.)

В зависимости от типа проекта конфигурационное тестирование может иметь разные цели:

- Проект по профилированию работы системы **Цель Тестирования**: определить оптимальную конфигурацию оборудования, обеспечивающую требуемые характеристики производительности и времени реакции тестируемой системы.
- Проект по миграции системы с одной платформы на другую **Цель Тестирования**: Проверить объект тестирования на совместимость с объявленным в спецификации оборудованием, операционными системами и программными продуктами третьих фирм.

Конфигурационное тестирование

Уровни проведения тестирования

Для клиент-серверных приложений конфигурационное тестирование можно условно разделить на два уровня (для некоторых типов приложений может быть актуален только один):

- Серверный
- Клиентский

На **первом (серверном)** уровне, тестируется взаимодействие выпускаемого программного обеспечения с окружением, в которое оно будет установлено:

- Аппаратные средства (тип и количество процессоров, объем памяти, характеристики сети / сетевых адаптеров и т.д.)
- Программные средства (ОС, драйвера и библиотеки, стороннее ПО, влияющее на работу приложения и т.д.)

Основной упор здесь делается на тестирование с целью определения оптимальной конфигурации оборудования, удовлетворяющего требуемым характеристикам качества.

Конфигурационное тестирование

На **следующем (клиентском)** уровне, программное обеспечение тестируется с позиции его конечного пользователя и конфигурации его рабочей станции. На этом этапе будут протестированы следующие характеристики: удобство использования, функциональность. Для этого необходимо будет провести ряд тестов с различными конфигурациями рабочих станций:

- Тип, версия и битность операционной системы (подобный вид тестирования называется **кросс-платформенное тестирование**)
- Тип и версия Web барузера, в случае если тестируется Web приложение (подобный вид тестирования называется **кросс-браузерное тестирование**)
- Тип и модель видео адаптера (при тестировании игр это очень важно)
- Работа приложения при различных разрешениях экрана
- Версии драйверов, библиотек и т.д. (для JAVA приложений версия JAVA машины очень важна, тоже можно сказать и для .NET приложений касательно версии .NET библиотеки)

Конфигурационное тестирование

Порядок проведения тестирования

Перед началом проведения конфигурационного тестирования рекомендуется:

- создавать матрицу покрытия (**матрица покрытия** - это таблица, в которую заносят все возможные конфигурации),
- проводить приоритезацию конфигураций (на практике, скорее всего, все желаемые конфигурации проверить не получится),
- шаг за шагом, в соответствии с расставленными приоритетами, проверяют каждую конфигурацию.

Связанные с изменениями виды тестирования

После проведения необходимых изменений, таких как исправление бага/дефекта, программное обеспечение должно быть перетестировано для подтверждения того факта, что проблема была действительно решена.

Виды тестирования, которые необходимо проводить после установки программного обеспечения, для подтверждения работоспособности приложения или правильности осуществленного исправления дефекта:

- **Дымовое тестирование**
- **Регрессионное тестирование**
- **Тестирование сборки**
- **Санитарное тестирование**

Дымовое тестирование

применяется для поверхностной проверки всех модулей приложения на предмет работоспособности и наличия быстро находимых критических и блокирующих дефектов.

Подвидом дымового тестирования являются тестирование сборки, выполняемые на функциональном уровне командой тестирования, по результатам которого делается вывод о том, принимается или нет установленная версия программного обеспечения в тестирование, эксплуатацию или на поставку заказчику.

Для облегчения работы, экономии времени и людских ресурсов рекомендуется внедрить автоматизацию тестовых сценариев для дымового тестирования.

Регрессионное тестирование

Регрессионное тестирование - это вид тестирования направленный на проверку изменений, сделанных в приложении или окружающей среде (починка дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб сервер или сервер приложения), для подтверждения того факта, что существующая ранее функциональность работает как и прежде.

Регрессионными могут быть как **функциональные**, так и **нефункциональные тесты**.

Как правило, **для регрессионного тестирования используются тест кейсы, написанные на ранних стадиях разработки и тестирования**. Это дает гарантию того, что изменения в новой версии приложения не повредили уже существующую функциональность.

Регрессионное тестирование

Сам по себе термин "Регрессионное тестирование", в зависимости от контекста использования может иметь разный смысл. Сэм Канер, к примеру, описал 3 основных типа регрессионного тестирования:

- **Регрессия багов** (**Bug regression**) - попытка доказать, что исправленная ошибка на самом деле не исправлена
- **Регрессия старых багов** (**Old bugs regression**) - попытка доказать, что недавнее изменение кода или данных сломало исправление старых ошибок, т.е. старые баги стали снова воспроизводиться.
- **Регрессия побочного эффекта** (**Side effect regression**) - попытка доказать, что недавнее изменение кода или данных сломало другие части разрабатываемого приложения

Тестирование сборки

- Тестирование направленное на определение соответствия, выпущенной версии, критериям качества для начала тестирования.
- По своим целям является аналогом дымового тестирования, направленного на приемку новой версии в дальнейшее тестирование или эксплуатацию.
- Вглубь оно может проникать дальше, в зависимости от требований к качеству выпущенной версии.

Санитарное тестирование

- **Санитарное тестирование** - это узконаправленное тестирование достаточное для доказательства того, что конкретная функция работает согласно заявленным в спецификации требованиям.
- Является подмножеством регрессионного тестирования.
- Используется для определения работоспособности определенной части приложения после изменений произведенных в ней или окружающей среде.
- Обычно выполняется вручную.
- В отличии от дымового, **санитарное тестирование направлено вглубь** проверяемой функции, в то время как **дымовое направлено вширь**, для покрытия тестами как можно большего функционала в кратчайшие сроки.

Уровни Тестирования Программного Обеспечения

Тестирование на разных уровнях производится на протяжении всего жизненного цикла разработки и сопровождения программного обеспечения.

Уровень тестирования определяет то, **над чем** производятся тесты: над отдельным модулем, группой модулей или системой, в целом.

1. Компонентное или Модульное тестирование (Component or Unit Testing)

- **Компонентное (модульное) тестирование проверяет функциональность и ищет дефекты в частях приложения**, которые доступны и могут быть протестированы по отдельности (**модули программ, объекты, классы, функции и т.д.**).
- Для этого вызывается код, который необходимо проверить и проверяется при поддержке сред разработки для модульного тестирования или инструментов для отладки. Все найденные дефекты, как правило исправляются в коде без формального их описания.

1. Компонентное или Модульное тестирование (Component or Unit Testing)

- Один из наиболее эффективных подходов к компонентному (модульному) тестированию - это **подготовка автоматизированных тестов** до начала основного кодирования (разработки) программного обеспечения. Это называется разработка от тестирования (**test-driven development**).
- При этом подходе создаются и интегрируются небольшие куски кода, напротив которых запускаются тесты, написанные до начала кодирования. Разработка ведется до тех пор пока все тесты не будут успешными.
- **Разница между компонентным и модульным тестированием в том, что** в компонентном тестировании в качестве параметров функций используют реальные объекты и драйверы, а в модульном тестировании - конкретные значения.

2. Интеграционное тестирование (Integration Testing)

Интеграционное тестирование предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами).

Уровни интеграционного тестирования:

Компонентный интеграционный уровень (*Component Integration testing*)

Проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.

Системный интеграционный уровень (*System Integration Testing*)

Проверяется взаимодействие между разными системами после проведения системного тестирования.

Подходы к интеграционному тестированию:

Снизу вверх (*Bottom Up Integration*) - Все низкоуровневые модули, процедуры или функции собираются воедино и затем тестируются. После чего собирается следующий уровень модулей для проведения интеграционного тестирования. Данный подход считается полезным, если все или практически все модули, разрабатываемого уровня, готовы. Также данный подход помогает определить по результатам тестирования уровень готовности приложения.

Сверху вниз (*Top Down Integration*) - Вначале тестируются все высокоуровневые модули, и постепенно один за другим добавляются низкоуровневые. Все модули более низкого уровня симулируются заглушками с аналогичной функциональностью, зачем по мере готовности они заменяются реальными активными компонентами. Таким образом мы проводим тестирование сверху вниз.

Большой взрыв (*"Big Bang" Integration*) - Все или практически все разработанные модули собираются вместе в виде законченной системы или ее основной части, и затем проводится интеграционное тестирование.

3. Системное тестирование (System Testing)

- Основной задачей системного тестирования является **проверка как функциональных, так и не функциональных требований в системе в целом**.
- При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д.
- Для минимизации рисков, связанных с особенностями поведения в системы в той или иной среде, **во время тестирования рекомендуется использовать окружение максимально приближенное к тому, на которое будет установлен продукт после выдачи**.

Два подхода к системному тестированию

- **на базе требований** (*requirements based*) - для каждого требования пишутся тестовые случаи (test cases), проверяющие выполнение данного требования.
- **на базе случаев использования** (*use case based*) - на основе представления о способах использования продукта создаются случаи использования системы (**Use Cases**). По конкретному случаю использования можно определить один или более сценариев. На проверку каждого сценария пишутся тест кейсы, которые должны быть протестированы.

4. Приемочное тестирование или Приемо-сдаточное испытание (Acceptance Testing)

Формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью:

- определения удовлетворяет ли система приемочным критериям;
- вынесения решения заказчиком или другим уполномоченным лицом принимается приложение или нет.

Приемочное тестирование **выполняется на основании набора типичных тестовых случаев и сценариев**, разработанных на основании требований к данному приложению.

Решение о проведении приемочного тестирования принимается, когда:

- продукт достиг необходимого уровня качества;
- заказчик ознакомлен с **Планом Приемочных Работ** или иным документом, где описан набор действий, связанных с проведением приемочного тестирования, дата проведения, ответственные и т.д.

Фаза приемочного тестирования длится до тех пор, пока заказчик не выносит решение об отправлении приложения на доработку или выдаче приложения.

Тестовые Артефакты

В соответствие с процессами или методологиями разработки ПО, во время проведения тестирования создается и используется определенное количество **тестовых артефактов** (документы, модели и т.д.). Наиболее распространенными тестовыми артефактами являются:

- **Тест План (План тестирования)**
- **Набор тест кейсов и тестов**
- **Баг (дефект) репорт.**

Тест План (План тестирования)

Тест план (**Test Plan**) - это документ, описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

Тест план – это документ, отвечающий на следующие вопросы:

- **Что надо тестировать?**
  - описание объекта тестирования: системы, приложения, оборудование
- **Что будете тестировать?**
  - список функций и описание тестируемой системы и её компонент в отдельности
- **Как будете тестировать?**
  - стратегия тестирования, а именно: виды тестирования и их применение по отношению к тестируемому объекту
- **Когда будете тестировать?**
  - последовательность проведения работ: подготовка (Test Preparation), тестирование (Testing), анализ результатов (Test Result Analisys) в разрезе запланированных фаз разработки

Тест План (План тестирования)

- **Критерии начала тестирования**:
  - готовность тестовой платформы (тестового стенда)
  - законченность разработки требуемого функционала
  - наличие всей необходимой документации
  - ...
- **Критерии окончания тестирования**:
  - результаты тестирования удовлетворяют критериям качества продукта
  - требования к количеству открытых багов выполнены
  - выдержка определенного периода без изменения исходного кода приложения **Code Freeze** (CF)
  - выдержка определенного периода без открытия новых багов **Zero Bug Bounce** (ZBB)
  - ...

Тест План (План тестирования)

Можно дополнить тест план следующими пунктами:

- Окружение тестируемой системы
- Необходимое для тестирования оборудование и программные средства
- Риски и их разрешение

Чаще всего на практике приходится сталкиваться со следующими видами тест планов:

- **Мастер Тест План (Master Plan or Master Test Plan)**
- **Тест План (Test Plan, назовем его детальный тест план)**
- **План Приемочных Испытаний (Product Acceptance Plan)** - документ, описывающий набор действий, связанных с приемочным тестированием: стратегия, дата проведения, ответственные работники и т.д.

Набор тест кейсов и тестов (Test Case & Test suite)

Тестовый случай (**Test Case**) - это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

Под тест кейсом понимается структура вида:

Action > **Expected Result** > **Test Result**

Тест кейсы разделяются по ожидаемому результату на **позитивные** и **негативные**:

- **Позитивный тест кейс** использует только корректные данные и проверяет, что приложение правильно выполнило вызываемую функцию.
- **Негативный тест кейс** оперирует как корректными так и некорректными данными (минимум 1 некорректный параметр) и ставит целью проверку исключительных ситуаций (срабатывание валидаторов), а также проверяет, что вызываемая приложением функция не выполняется при срабатывании валидатора.

Пример Тест кейса

Название:	Тест отправки комментария	
Функция:	Контакт-Вопросы	
Действие	**Ожидаемый результат**	**Результат теста:** • пройден • провален • заблокирован
Предусловие:		
Откройте сайт Про Тестинг: http://www.protesting.ru	Сайт Про тестинг открыт и доступен	
Перейдите по ссылке "Задать вопрос" внизу страницы	Страница "Вопросы, пожелания и заявки" открыта и доступна	
Шаги теста:		
Заполните форму отправки комментария: "Тип Обращения": Комментарий "Контактное лицо": Ольга "E-mail": test@test.com "Сообщение": *Добрый день, уважаемый коллектив "ПроТестинг"!* *Я еще ни разу не видела кач. примера баг-репорта, тест-кейса и прочей необходимой док-ии.* *Не подскажите, где я могу с ними ознакомиться?* *С уважением.*	Данные успешно введены	
Нажмите кнопку "Отправить"	Страница "Ваш запрос успешно отправлен!" открыта	
Постусловие:		
Кликните по ссылке "Перейти Назад на форму отправления заявок"	Страниц "Вопросы, пожелания и заявки" открыта	

Структура Тестовых Случаев (Test Case Structure)

Один из подходов к проектированию тест кейсов - каждый тест кейс должен иметь 3 части:

PreConditions	Список действий, которые приводят систему к состоянию пригодному для проведения основной проверки. Либо список условий, выполнение которых говорит о том, что система находится в пригодном для проведения основного теста состояния.
Test Case Description	Список действий, переводящих систему из одного состояния в другое, для получения результата, на основании которого можно сделать вывод о удовлетворении реализации, поставленным требованиям
PostConditions	Список действий, переводящих систему в первоначальное состояние (состояние до проведения теста - initial state)

Примечание: **Post Conditions** не является обязательной частью. Это скорее всего - правило хорошего тона: "намусорил - убери за собой". Это особенно актуально при автоматизированном тестировании, когда за один прогон можно наполнить базу данных сотней или даже тысячей некорректных документов.

Детализация описания тест кейсов

Пример тест кейса 1:

Проверка отображения страницы		
Действие	**Ожидаемый результат**	**Результат теста**
Открыть страницу "Вход в систему"	- Окно "Вход в систему" открыто - Название окна - Вход в систему - Логотип компании отображается в правом верхнем углу - На форме 2 поля - Имя и Пароль - Кнопка Вход доступна - Ссылка "забыл пароль" - доступна	...

Детализация описания тест кейсов

Пример тест кейса 2:

Название: Проверка отображения страницы
Действие: Открыть страницу "Вход в систему"
Проверка: Проверьте, что отображаемая страница соответствует странице на картинке 1 (и прилагаем изображение страницы "Вход в систему")

Обычно считается, что **уровень детализации тест кейсов** должен быть таков, чтобы обеспечивать разумное соотношение времени прохождения к тестовому покрытию. Т.е. до тех пор пока покрытие тестами определенного функционала не меняется, можно уменьшать детализацию тест кейсов.

В примере 1 и 2 покрытие будет одинаковым, но вот время, которое потребуется для прохождения, будет разным. Второй пример будет даже нагляднее.

Баг Репорт (Bug Report)

Баг или дефект репорт - это документ, описывающий ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

Структура баг репорта

- Разные системы менеджмента дефектами, предлагают нам разные поля для заполнения и разные структуры описания дефектов. Нижеприведенная таблица – это один из вариантов **шаблона баг репорта**.

Баг Репорт (Bug Report)

Шапка	
Короткое описание (Summary)	Короткое описание проблемы, явно указывающее на причину и тип ошибочной ситуации.
Проект (Project)	Название тестируемого проекта
Компонент приложения (Component)	Название части или функции тестируемого продукта
Номер версии (Version)	Версия на которой была найдена ошибка
Серьезность (Severity)	Наиболее распространена пятиуровневая система градации серьезности дефекта: • S1 Блокирующий (Blocker) • S2 Критический (Critical) • S3 Значительный (Major) • S4 Незначительный (Minor) • S5 Тривиальный (Trivial)
Приоритет (Priority)	Приоритет дефекта: • P1 Высокий (High) • P2 Средний (Medium) • P3 Низкий (Low)

Баг Репорт (Bug Report) продолжение таблицы

Статус (Status)	Статус бага. Зависит от используемой процедуры и жизненного цикла бага (bug workflow and life cycle)
Автор (Author)	Создатель баг репорта
Назначен на (Assigned To)	Имя сотрудника, назначенного на решение проблемы
Окружение	
ОС / Сервис Пак и т.д. / Браузера + версия / …	Информация об окружении, на котором был найден баг: операционная система, сервис пак, для WEB тестирования - имя и версия браузера и т.д.
…	
Описание	
Шаги воспроизведения (Steps to Reproduce)	Шаги, по которым можно легко воспроизвести ситуацию, приведшую к ошибке.
Фактический Результат (Result)	Результат, полученный после прохождения шагов к воспроизведению
Ожидаемый результат (Expected Result)	Ожидаемый правильный результат
Дополнения	
Прикрепленный файл (Attachment)	Файл с логами, скриншот или любой другой документ, который может помочь прояснить причину ошибки или указать на способ решения проблемы

Тест Дизайн (Test Design)

Тест дизайн – это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест кейсы), в соответствии с определёнными ранее критериями качества и целями тестирования.

План работы над тест дизайном

- анализ имеющихся проектных артефактов: документация (спецификации, требования, планы), модели, исполняемый код и т.д.
- написание спецификации по тест дизайну (Test Design Specification)
- проектирование и создание тестовых случаев (Test Cases)

Роли, ответственные за тест дизайн

- Тест аналитик - определяет "**ЧТО тестировать?**"
- Тест дизайнер - определяет "**КАК тестировать?**"

Тест Дизайн (Test Design)

Задача тест аналитиков и дизайнеров сводится к тому, чтобы используя различные стратегии и техники тест дизайна, создать набор тестовых случаев, обеспечивающий оптимальное тестовое покрытие тестируемого приложения.

- **Тестовое Покрытие** (**Test Coverage**) - это одна из метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.
- **Детализация Тест Кейсов** (**Test Case Specification**) - это уровень детализации описания тестовых шагов и требуемого результата, при котором обеспечивается разумное соотношение времени прохождения к тестовому покрытию
- **Время Прохождения Тест Кейса** (**Test Case Pass Time**) - это время от начала прохождения шагов тест кейса до получения результата теста.

Процесс тестирования программного обеспечения

Тестирование ПО начинается с момента начала работы над проектом.

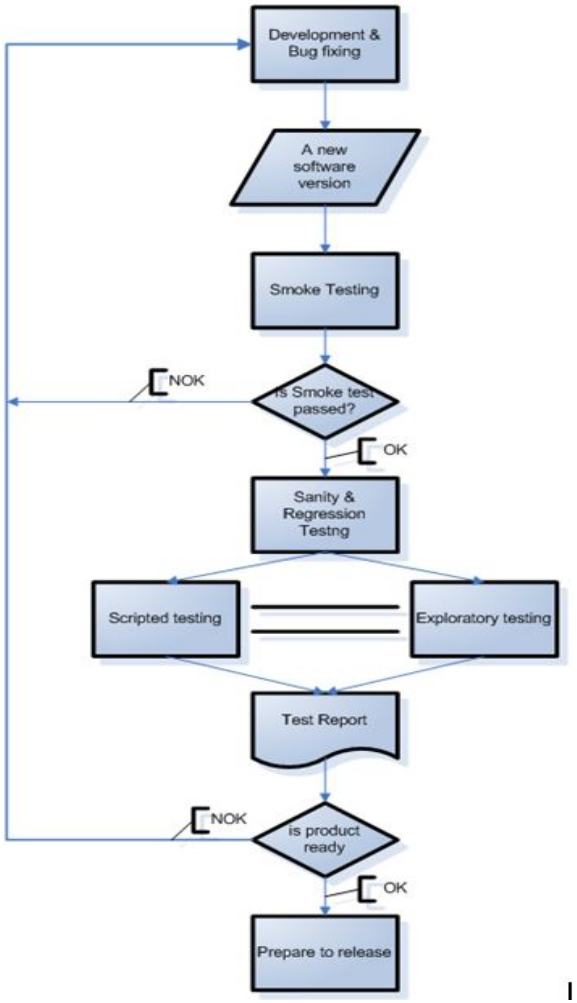
После получения первых спецификаций можно писать тест план, затем разрабатывать тест кейсы, оценивать необходимость использования автоматизации, причем как автоматизации ***функционального тестирования***, так и ***нагрузочного***.

Как только разработчики подготовили версию, проводится ***дымовое тестирование***, по результатам которого делается вывод о возможности и целесообразности дальнейшего тестирования:

В случае если "**smoke test failed**!!!" - приложение на доработку.

Если же "**smoke test passed**!!!« - переход к следующим видам тестирования - ***регрессионное тестирование*** (**Regression testing**) и ***санитарное тестирование*** (**Sanity testing**).

Процесс тестирования программного обеспечения



- Открыв багтрекер, нужно перепроверить **дефекты**, которые разработчики перевели в статус Fixed (Исправлено), Rejected, Can't Reproduce и т.д.
- Затем – централизованное тестирование по тест кейсам и/или "исследование" приложения.
- Результат – результаты прогона тест кейсов, баг репорты, вопросы к аналитикам.
- Составляется отчет по проведенному тестированию и отправляется на проектную группу.
- Подобный процесс проходит от версии к версии, и через какое-то время результаты тестирования сойдутся, с прописанными в плане тестирования критериями окончания тестирования.

Автоматизированное тестирование

Средства автоматизированного тестирования:

- Функциональное тестирование – Mercury QuickTest Professional, Mercury WinRunner, IBM Rational Robot/TestManager, IBM Rational Functional Tester;
- Нагрузочное тестирование – Mercury LoadRunner, IBM Rational Robot/TestManager, IBM Rational Performance Tester.