Виды Тестирования Программного Обеспечения

BitPaper
Share Knowledge

Функциональные

Функциональные тесты базируются на функциях и особенностях, а также взаимодействии с другими системами. Функциональные виды тестирования рассматривают внешнее поведение системы.

1. Функциональное тестирование (Functional testing)
2. Тестирование безопасности (Security and Access Control Testing)
3. Тестирование взаимодействия (Interoperability Testing)

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 случаев использования системы (use cases).

Тестирование взаимодействия (Interoperability Testing) – 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).

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

Связанные с изменениями

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

1. Дымовое тестирование (Smoke Testing)
2. Регрессионное тестирование (Regression Testing)
3. Тестирование сборки (Build Verification Test)
4. Санитарное тестирование или проверка согласованности/исправности (Sanity Testing)

Дымовое тестирование - короткий цикл тестов, выполняемый для подтверждения того, что после сборки кода (нового или исправленного) устанавливаемое приложение, стартует и выполняет основные функции

Регрессионное тестирование - проверка изменений, сделанных в приложении или окружающей среде, для подтверждения факта, что существующая ранее функциональность работает как и прежде

Тестирование сборки - определение соответствия, выпущенной версии, критериям качества для начала тестирования. По своим целям является аналогом Дымового Тестирования

Санитарное тестирование - узконаправленное тестирование достаточное для доказательства того, что конкретная функция работает согласно заявленным в спецификации требованиям.

Используется для определения работоспособности определенной части приложения после изменений произведенных в ней или окружающей среде.
Обычно выполняется вручную

Нефункциональные

Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. В целом,
это тестирование того, "Как" система работает.

1. Тестирование установки (Installation testing)
2. Конфигурационное тестирование (Configuration Testing)
3. Тестирование удобства пользования (Usability Testing)
4. Тестирование на отказ и восстановление (Failover and Recovery Testing)
5. Все виды тестирования производительности:
   + нагрузочное (Performance and Load Testing)
   + стрессовое (Stress Testing)
   + стабильности или надежности (Stability / Reliability Testing)
   + объемное (Volume Testing)
6. Тестирование безопасности (Security and Access Control Testing)

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

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

Уровни Тестирования Программного Обеспечения

BitPaper
Share Knowledge

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.

Проведение тестирования на всех уровнях системы - это залог успешной реализации и сдачи проекта.

Уровни Тестирования
- Компонентное или Модульное (Component Testing or Unit Testing)
- Интеграционное (Integration Testing)
- Системное (System Testing)
- Приемочное (Acceptance Testing)

Уровни Тестирования Программного Обеспечения

Компонентное или Модульное (Component Testing or Unit Testing)

проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули программ, объекты, классы, функции и т.д.)

Интеграционное (Integration Testing)

- проверка связи между компонентами, а также
- взаимодействие с различными частями системы (операционной системой, оборудованием либо связи между различными системами)

Подуровни:

- 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)
+ Сверху вниз (Top Down Integration)
+ Большой взрыв ("Big Bang" Integration)

Системное (System Testing)

проверка как функциональных, так и не функциональных требований в системе в целом

Подходы к системному тестированию:

+ на базе требований (requirements based)
+ на базе случаев использования (use case based)

Приемочное (Acceptance Testing)

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.

BitPaper
Share Knowledge

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.).

Разница между компонентным и модульным тестированием

По-существу эти уровни тестирования представляют одно и тоже, разница лишь в том, что
в компонентном тестировании в качестве параметров функций используют реальные объекты и драйверы,
а в модульном тестировании - конкретные значения.

Интеграционное тестирование (Integration Testing)

BitPaper
Share Knowledge

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 (см. также Integration testing - Bottom Up)

+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, затем по мере готовности они заменяются реальными активными компонентами.
Таким образом мы проводим тестирование сверху вниз. (см. также Top Down Integration)

+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. Такой подход очень хорош для сохранения времени.
Однако если тест кейсы и их результаты записаны не верно, то сам процесс интеграции сильно осложнится, что станет преградой для команды тестирования при достижении основной цели интеграционного тестирования (см. также Integration testing - Big Bang)

Автоматизированное тестирование программного обеспечения - основные понятия

BitPaper
Share Knowledge

Автоматизированное тестирование программного обеспечения (Software Automation Testing) - это процесс верификации программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, выполняются автоматически при помощи инструментов для автоматизированного тестирования.

Специалист по автоматизированному тестированию программного обеспечения (Software Automation Tester) - это технический специалист (тестировщик или разработчик программного обеспечения), обеспечивающий создание, отладку и поддержку работоспособного состояния тест скриптов, тестовых наборов и инструментов для автоматизированного тестирования.

Инструмент для автоматизированного тестирования (Automation Test Tool) - это программное обеспечение, посредством которого специалист по автоматизированному тестированию осуществляет создание, отладку, выполнение и анализ результатов прогона тест скриптов.

Тест Скрипт (Test Script) - это набор инструкций, для автоматической проверки определенной части программного обеспечения.

Тестовый набор (Test Suite) - это комбинация тест скриптов, для проверки определенной части программного обеспечения, объединенной общей функциональностью или целями, преследуемыми запуском данного набора.

Тесты для запуска (Test Run) - это комбинация тест скриптов или тестовых наборов для последующего совместного запуска (последовательного или параллельного, в зависимости от преследуемых целей и возможностей инструмента для автоматизированного тестирования).

Юнит-тесты (вики) — это тестирование одного элемента кода (например, отдельная функция или класс в случае ООП-кода) в изоляции от остальной части программы. Это значит, что если код обращается к каким-то другим классам, то вместо них подсовываются классы-заглушки (моки и стабы). Если код обращается к файлам, базе данных, по сети, то это все тоже заменяется на заглушки, возвращающие заранее подготовленные данные.

Стабы — это классы-заглушки, которые вместо выполнения действия возвращают какие-то данные. Например, стаб класса работы с базой данных может вместо реального обращения к базе данных возвращать, что запрос успешно выполнен. А при попытке прочитать что-то из нее возвращает готовый массив с данными.

Моки — это классы-заглушки, которые используются чтобы проверить, что определенная функция была вызвана с определенными аргументами.

BitPaper
Share Knowledge

BitPaper
Share Knowledge

Тест-кейс — это проверка.
"Выполни тест-кейс по вводу отрицательных значений" = проведи проверку такую-то и проверь, что результат будет такой-то.

Тест-кейс — это такое описание проверки работы системы, которое может выполнить любой человек из команды, будь то тестировщик, разработчик, аналитик или даже бизнес-заказчик.

Набор тест-кейсов называется тестовым набором (test suite).
Иногда этот набор некорректно называют тест-планом.
Тест-план — это именно план: когда, что, зачем, какими ресурсами.

Стандартные атрибуты тест-кейса

+ Номер — уникальный идентификатор тест-кейса.
  Его удобно использовать для одинакового понимания,
  о какой проверке идет речь (например, дать ссылку в баге).

+ Название — краткое описание сути проверки.
  Должно помещаться в твиттер и быть понятным! Кратко, но емко.

+ Предварительные шаги — описание действий, которые необходимо выполнить,
  но прямого отношения к проверке они не имеют
  (например, зарегистрироваться в системе для проверки создания элемента).
  Если предварительных шагов нет, то секция не заполняется.

+ Шаги — описание действий, необходимых для проверки
  (например, создание элемента).

+ Ожидаемый результат (ОР) — сама проверка: что мы ожидаем получить после выполнения шагов
  ("Элемент создан").

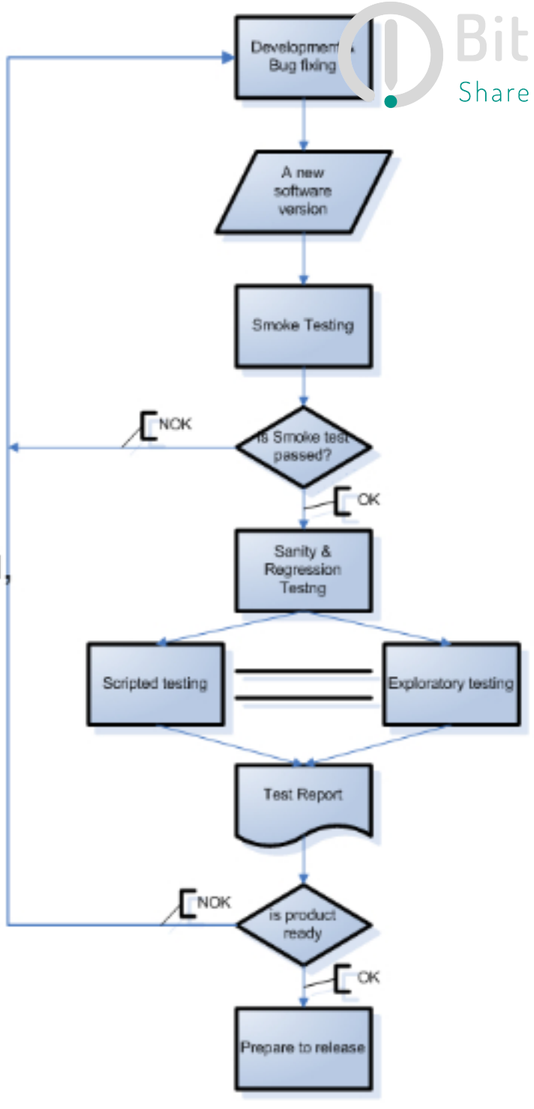
Тестовая документация

- тест план
- тест кейсы
- багтрекер
- результаты прогона тест кейсов
- чек-лист
- баг репорты
- отчет по проведенному тестированию

+ План тестирования (Test Plan)
  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.

+ Набор тест кейсов и тестов (Test Case & Test suite)
  последовательность действий, по которой можно проверить
  соответствует ли тестируемая функция установленным требованиям.

+ Дефекты / Баг Репорты (Bug Reports / Defects)
  документы, описывающие ситуацию или последовательность действий
  приведшую к некорректной работе объекта тестирования,
  с указанием причин и ожидаемого результата.



BitPaper
Share Knowledge

Типы тестирования

Этапы тестирования

1. Подготовка
2. Непосредственно тестирование
3. Анализ результатов и составление отчетов

BitPaper
Share Knowledge

Техники тест-дизайна

BitPaper
Share Knowledge

Граничные значения

большинство ошибок
может возникнуть на границах
эквивалентных классов

\+

Эквивалентное разбиение

позволяет минимизировать число тестов,
не создавая сценарий для каждого
возможного значения, а выбрав
только одно значение из целого класса

Попарное тестирование

(pairwise testing)
каждое значение каждого
проверяемого параметра
должно быть протестировано
на взаимодействие с каждым значением
всех остальных параметров

Убираем тесты,
которые дублируют друг друга,
оставляя максимальное покрытие
при минимальном необходимом
наборе сценариев

Google Chrome	Windows	RU
Google Chrome	Windows	EN
Google Chrome	Linux	RU
Google Chrome	Linux	EN
Opera	Windows	RU
Opera	Windows	EN
Opera	Linux	RU
Opera	Linux	EN

Google Chrome	Windows	RU
Google Chrome	Linux	EN
Opera	Windows	EN
Opera	Linux	RU

Исчерпывающее тестирование

Таблица принятия решений

(матрица принятия решений)
пример:
двухфакторная аутентификация

Причина и следствие

Простая проверка
базовых действий
и их результата

Примерный алгоритм использования техники:

1. Выделяем причины и следствия в спецификациях.
2. Связываем причины и следствия.
3. Учитываем «невозможные» сочетания причин и следствий.
4. Составляем «таблицу решений», где в каждом столбце
   указана комбинация входов и выходов, т.е.
   каждый столбец – это готовый тестовый сценарий.
5. Расставляем приоритеты.

Эта техника помогает:

- Определить минимальное количество тестов
  для нахождения максимума ошибок.

- Выяснить все причины и следствия – таким образом,
  мы убедимся, что на любые манипуляции с системой
  у системы будет ответ.

- Найти возможные недочеты в логике описания приложения
  (что, в свою очередь, поможет улучшить документацию).

Предугадывание ошибок

Тестер опытен,
хорошо знает продукт
и общается с коллегами

Недостатки:
1. Техника в значительной степени
   основана на интуиции.
2. Необходим опыт в тестировании
   подобных систем.
3. Малое покрытие тестами.