Урок №2

Типы тестирования. Документооборот в процессе тестирования.

Содержание

1. Типы тестирования
    a. Unit Testing
    b. Integration Testing
    c. System Testing
    d. Functional Testing
    e. Regression Testing
    f. Smoke Testing
    g. GUI Testing
    h. Load testing
    i. Performance Testing
    j. Stress Testing
    k. Security Testing
    l. Installation Testing
    m. Platform/Environment Testing
    n. Parallel/Comparison Testing
    o. Usability Testing
    p. Conformance Testing
    q. User Acceptance/Alpha/Beta Testing
    r. Другие типы тестирования

2. Цели и задачи документации в процессе тестирования
3. План тестирования (Test Plan)
    a. Что такое план тестирования
    b. Цели и задачи плана тестирования
    c. Когда необходимо разрабатывать план тестирования?
    d. Шаблон плана тестирования в формате IEEE 829
    e. Разделы плана тестирования
        1. Test Plan Identifier, Authors and Revision History
        2. References
        3. Introduction
        4. Software Risk Issues
        5. Test Items
        6. Features to be Tested
        7. Features not to be Tested
        8. Approach
        9. Item Pass/Fail Criteria
        10. Suspension Criteria and Resumption Requirements
        11. Test Deliverables
        12. Environmental Needs
        13. Staffing and Training Needs
        14. Responsibilities
        15. Schedule
        16. Planning Risks and Contingencies
        17. Approvals
        18. Glossary
    f. Пример создания плана тестирования
4. Test Design
    a. Что такое Test Design?
    b. Виды документов для тестирования
        1. Checklist
        2. Test Case

3. Test Scenario
4. Test Suite/Test Design Specification/Test spec

c. Примеры создания документов (Checklist, Test Case, Test Scenario,…)

5. Отчеты об ошибках (Bug Reporting)
   a. Что такое отчет об ошибках?
   b. Жизненный цикл отчета
   c. Формат отчета
   d. Примеры отчетов об ошибках
   e. Системы отслеживания ошибок (bugtracking systems).

1. Типы тестирования

Зачем необходимо классифицировать методы тестирования? Зачем вообще необходима типизация? Классификация, как базовая аналитическая операция позволяет выделить специфичные, важные черты объекта (в нашем случае метода) и отсеять всё второстепенное, что, в контексте классификации методов, позволяет правильно применять последние. Иначе говоря, применять метод там, де это нужно.

С другой стороны, в силу сложности программного обеспечения, полное тестирование предполагает исследование всех сторон и качеств тестируемого программного продукта. А обязательное стремление тестирования программного обеспечения к полноте, в свою очередь, неизбежно порождает различные методы и подходы в тестировании; поскольку очевидно, что способы и приёмы, применяемые для тестирования, например, интерфейса, не будут эффективны (показательны) при инсталляционном (установочном) тестировании. И знать классификацию методов необходимо хотя бы для того, чтобы впоследствии применять их уместно и по назначению.

В предыдущем уроке уже упоминалось о типах тестирования. Далее же последует их более подробное описание.

Unit Testing

Unit testing или *модульное тестирование* – это метод тестирования, позволяющий удостовериться в том, что каждая наименьшая неделимая часть программы (модуль) работает должным образом в изолированном состоянии. Другими словами, цель модульного тестирования состоит в том, чтобы отделить отдельные части кода (программы) и убедиться, что они работают правильно (так, как предполагается).

Модуль определяется произвольным образом – им может быть класс или библиотека. Хотя классически в модульном тестировании речь идёт о исполняемом коде и применяется оно на уровне «компонентного и модульного тестирования», его принципы переносятся и на другие уровни. На различных уровнях приближения в качестве наименьшего элемента будут выступать различные лексические единицы. Так, в процедурном подходе наименьшим неделимым элементом есть функция, которая представляет собой реализацию некоторого законченного алгоритма. Тогда как в объектно-ориентированном подходе таким элементом будет метод класса.

Из преимуществ, которые предоставляет модульное тестирование можно выделить:

- Облегчение изменения кода, поскольку разработчик получает возможность внести изменения в код постфактум, при сохранении функциональности модуля. Например, в ситуации, когда при проведении модульного тестирования обнаружилось, что метод генерации псевдослучайных чисел выполняется за более длительное время, нежели это предполагалось спецификацией;
- Упрощение интеграции, которое происходит за счёт того, что модульное тестирование устраняет неопределённость, поскольку вначале тестируются отдельные модули, а затем результат их совместной работы; в таком смысле модульное тестирование используют в так называемом тестировании снизу-вверх;

- Поддержка документирования состоит в том, что разработчики могут посмотреть на описание того, как функционирует модуль, а так же могут изучить результаты тестирования, для более полного понимания того, как именно функционирует модуль.

Integration Testing

Интеграционное тестирование (integration testing) – это метод тестирования, который исследует взаимодействие между двумя или более компонентами приложения. В более широком понимании, интеграционное тестирование ставит целью проверить, соответствует ли взаимодействие модулей приложения критериям функциональности, производительности и надёжности, определённым спецификациями.

Проходит интеграционное тестирование, как тестирование группы модулей посредством взаимодействия с интерфейсом получившейся группы. Как правило, в интеграционном тестировании используется метод «тестирования чёрного ящика». Удачные и неудачные тестовые ситуации симулируются посредством подачи на входы группы соответствующих тестовых данных.

Ошибки интегрирования могут происходить по следующим причинам:

- неправильное использование интерфейса (interface misuse) – вызываемый модулем компонент вызывает ещё один компонент, однако, использует его ошибочно (например, данные в вызываемый метод передаются в неправильной последовательности), что вызывает ошибку функционирования всего комплекса;
- ошибочное понимание интерфейса (interface misunderstanding) – тестируемый компонент делает некоторое предположение о поведении другого, вызываемого им, компонента, которое является ошибочным.

Различают следующие виды интеграционного тестирования:

- интеграционное тестирование по принципу «Большого Взрыва» (Big Bang Integration Testing) – в рамках данного подхода к интеграционному тестированию все, или большинство, модулей, системы собираются вместе в виде полного приложения или его основной части, и потом подвергаются интеграционному тестированию. Использую интеграционное тестирование по принципу «Большого Взрыва» команда тестировщиков может сэкономить время. Однако, если тестовые ситуации и их результаты были зафиксированы неверно, цель интеграционного тестирования может быть не достигнута, поскольку увеличивается опасность пропустить возможные тестовые ситуации при планировании, в силу сложности тестируемой системы (чем модуль меньше и проще, тем выше меньше количество тестовых ситуаций, а значит выше шансы достигнуть полного тестового покрытия), а также возникает возможность влияния одних компонент приложения на функционирование других, что может привести к появлению неочевидных «багов».
- интеграционное тестирование снизу вверх (Bottom UP Integration Testing) – это подвид интеграционного тестирования, который предполагает, что низкоуровневые модули тестируются до того, как будут использованы для тестирования более высокоуровневых модулей. Процесс тестирования, в данном подходе, повторяется до тех пор, пока не будет достигнута вершина системы иерархи модулей.

  Так как, при использовании данного подхода, высокоуровневые модули не всегда готовы к началу тестирования более низкоуровневых, то для их тестирования используется вспомогательное программное обеспечение, которое подаёт на входы модулей тестовые данные, и тем самым симулирует поведение модуля, интегрирующего тестируемый.

  Использование интеграционного тестирования снизу вверх даёт уверенность в том, что высокоуровневые модули будет

протестированы правильно, поскольку на момент их тестирования уже закончено тестирования компонент, которые ими используются.

Отрицательные стороны данного подхода состоят в том, что он не применим в проектах, которые используют способ разработки «сверху вниз», а также в том, что создание симуляторов, используемых данным подходом, достаточно сложно и требует дополнительных производственных мощностей (а именно человеческих ресурсов).

- интеграционное тестирование сверху вниз (Top Down Integration Testing) – подвид интеграционного тестирования, в котором высокоуровневые модули тестируются в первую очередь, а вызываемые ими низкоуровневые модули симулируются некоторыми программными средствами (возможно, псевдо модулями), подражающими их поведению. Преимущество состоит в том, что интеграционное тестирование можно начинать до того, как будут закончены все модули приложения, а также появляется возможность создавать код асинхронно. Другими словами, программист может создавать высокоуровневые модули ещё до того, как будет написан низкоуровневый код.

  Например, программа использует библиотеку, заказанную у стороннего разработчика, и она не готова к тому моменту, как уже закончена разработка ядра системы. Обычно, приходиться ждать завершения всех модулей, для того чтобы начать интеграционное тестирование. Данный же подход позволяет начинать тестирование на ранних этапах разработки, что позволяет выявлять дефекты архитектуры раньше, чем обычно, и, следовательно, экономит время и ресурсы.

  Отрицательной стороной данного подхода является то, что программные средства, имитирующие необходимую приложению функциональность низкого уровня тоже необходимо кому-то разрабатывать. Конечно, времени тратиться меньше, нежели на

разработку реального модуля, однако зачастую на это просто не хватает ресурсов (либо рабочего времени программистов, либо бюджета проекта для того, чтобы нанять программистов специально для этой задачи).

- гибридное интеграционное тестирование (Hybrid integration Testing) - описанные выше подходы (речь идёт об интеграционном тестировании сверху вниз и интеграционном тестировании снизу вверх), как было нами указано, имеют как положительные, так и отрицательные стороны. Гибридное же интеграционное тестирование (его ещё называют тестированием по принципу сандвича – sandwich testing) предлагает скомбинировать данные подходы таким образом, чтобы получить максимальный эффект. Иначе говоря, применять каждый метод в там, где он даёт максимальную пользу. Но, необходимо помнить, что нужно тщательно планировать и фиксировать тестовые операции, поскольку может возникнуть путаница: какой из модулей тестировался «сверху вниз», а какой «снизу вверх».

System Testing

Системное тестирование (system testing) – одна из фаз полного цикла тестирования программного обеспечения, целью которой является оценка соответствия системы выдвигаемым к ней требования (спецификациям), а также подтверждение того, что отсутствуют несоответствия между интегрированным в рамках системы программным или аппаратным обеспечением.

Проводиться системное тестирование тогда, когда вся система уже закончена и все её компоненты успешно интегрированы. Соответственно, проводиться системное тестирование только тогда, когда пройдены модульное, функциональное, интеграционное и компонентное тестирование.

Необходимо также отметить, что системное тестирование должно проводиться по принципу «чёрного ящик», то есть тестировщик должен воспринимать тестируемую систему, как заведомо неизвестную. Отсутствие знаний о системе делает тестировщика объективным.

И подходить к системному тестированию следует именно таким образом – условно забыв всё, что нам о ней известно. Некоторые подходы, такие как модульное тестирование, невозможны без знания структуры и даже исходного кода, тогда как в системном тестировании они могут навредить.

Системное тестирование – наиболее важный этап всего жизненного цикла тестирования и предполагает проведение таких видов тестирования как:

- тестирование пользовательского интерфейса (GUI software testing) – вид тестирования, который применяется к программным продуктам, использующим графический пользовательский интерфейс (GUI), с целью установить, что GUI отвечает заявленным спецификациям;
- тестирование удобства использования (Usability testing) – вид тестирования, целью которого является оценка того, насколько программный продукт удобен в использовании;
- тестирование производительности (Performance testing) – вид тестирования программного обеспечения направленный на оценку того, соответствует ли функционирование системы при штанных нагрузках заявленным в спецификациях характеристикам;
- тестирование совместимости (Compatibility testing) – вид тестирования призванный удостовериться в том, что разработанная система совместима с окружением, в котором ей предстоит функционировать;
- тестирование обработки ошибок (Error handling testing) – вид тестирования программного обеспечения, целью которого является проверка того, что система корректно обрабатывает исключительные ситуации, возникающие во время функционирования;
- нагрузочное тестирование (Load testing) – вид тестирования программного обеспечения, назначение которого состоит в том, чтобы

удостовериться в том, что система будет функционировать нормально при ожидаемых штатных и пиковых нагрузках;

- объёмное тестирование (Volume testing) – вид тестирования, который заключается в том, чтобы проверить сможет ли приложение обрабатывать тот объём данных, который предполагается для обработки данной системой в штатном состоянии в соответствии с заявленными спецификациями;
- стрессовое тестирование (Stress testing) – вид тестирования, который предназначен для оценки стабильности системы. Стрессовое тестирование состоит в том, чтобы изучить поведение системы в условиях нагрузок, превышающих штатные;
- тестирование безопасности (Security testing) – вид тестирования программного обеспечения, который состоит в том, чтобы проверить, что информационная система защищает данные от несанкционированного проникновения;
- тестирование масштабируемости (Scalability testing) – вид тестирования программного обеспечения, направленный на измерение масштабируемости программного обеспечения, под которой понимается возможность расширения или сужения нефункциональных характеристик последнего;
- тестирование вменяемости (Sanity testing) – вид тестирования направленный на то, чтобы быстро выяснить – существует ли вероятность того, что результат функционирования программы является корректным;
- дымовое тестирование (Smoke testing) - это комплекс тестов, которые направлены на подтверждение (опровержение) того, что основные, наиболее важные, функции программного продукта выполняются должным образом. При этом не акцентируется внимание на несущественных деталях, а имеются в виду только наиболее важные функции;

- исследовательское тестирование (Exploratory testing) – это подход к тестированию программного обеспечения, который предполагает тестирование, как процесс одновременного изучения тестируемого объекта, разработку и проведение теста;
- регрессионное тестирование (Regression testing) – это подход к тестированию, который направлен на то, чтобы удостовериться, что процесс исправления ошибок не внёс дополнительных (новых) ошибок.
- тестирование надёжности (Reliability testing) – это вид тестирования, который изучает поведение системы под воздействием штатных нагрузок на протяжении длительного функционирования. Целью теста является подтвердить, что основные (важные с точки зрения функционирования) характеристики системы изменяются в допустимых заявленными спецификациями диапазонах при длительном функционировании.
- инсталляционное тестирование (Installation testing) – вид тестирования, направленный на изучение процесса установки программного продукта и удостоверение того, что процесс инсталляции и возможности, которые он предполагает, функционируют должным образом и удовлетворят потребности конечного пользователя.
- тестирование отказов и восстановления (Recovery testing and failover testing) – вид тестирования призванный определить насколько система способна к восстановлению после отказов.
- тестирование доступности (Accessibility testing) – данный вид тестирования направлен на то, чтобы удостовериться, что система обладает средствами, позволяющими людям с ограниченными возможностями (инвалидам) взаимодействовать с системой. Данное тестирование имеет место в том случае, если система обладает пользовательским интерфейсом.

Functional Testing

Функциональное тестирование (functional testing) – это вид тестирования программного обеспечения, который концентрируется на исследовании функций выполняемых тестируемым программным продуктом. Прежде всего, тестирование должно подтвердить, что разработанная система действительно выполняет заявленные спецификациями функции, а так же, что эти функции выполняются так, как предусмотрено. Хотя, обычно, первое подразумевает и второе.

В целях формализации процесса тестирования принято говорить, что функциональное тестирование оценивает реализуемость функциональных требований, которые включают:

- функциональную пригодность (suitability);
- точность (accuracy);
- способность к взаимодействию (interoperability);
- соответствие стандартам и правилам (compliance);
- защищённость (security).

Очевидно, что для оценки безопасности и взаимодействия используют, соответственно, тестирование безопасности (security testing) и взаимодействия (interoperability testing). Остальные же функциональные требования оцениваются, собственно, функциональным тестированием.

Так как, функциональное тестирование оценивает функциональные требования к программному обеспечению, то его основное внимание сосредотачивается на исследовании поведения системы, а не особенностей её реализации.

Regression Testing

Как уже не раз упоминалось, регрессионное тестирование (regression testing) предназначено для выявления ошибок связанных с устранение ошибок, найденных на более ранних стадиях тестирования. В таком смысле его уместнее назвать не регрессионным тестирование, а тестированием «на регрессию».

Также регрессионное тестирование можно понимать, как тестирование на предмет выявления потери функциональности в связи с внесёнными в код изменениями. Причина этих изменений не обязательно должна состоять в исправлении найденных ошибок – это может быть добавление к системе некоторого дополнения, расширяющего его функциональность, и вообще любое изменение.

Появление ошибок при изменении кода явление не такое редкое, каким может показаться на первый взгляд. При этом совершенно не обязательно, что ошибка будет состоять в нарушении функционирования самого изменяемого модуля. Ошибка может проявиться, как конфликт в смысле взаимодействия, поскольку были внесены некоторые связи, отсутствовавшие в предыдущей реализации и повлекшие за собой несовместимость. И вариаций можно придумать множество.

На сегодняшний день достаточно широко используется такой подход к разработке программного обеспечения как итерационная разработка. В рамках этого подхода используются короткие временные отрезки в качестве базовых периодов разработки. И предполагается, что на выходе каждого цикла мы должны получать добавление новой функциональности к приложению. Было бы целесообразным, в конце каждого цикла поверять, не привели ли внесённые дополнения к потере базовой (то, что была до изменений) функциональности.

Smoke Testing

Дымовое тестирование (smoke testing) – это набор из тестовых ситуаций, которые покрывают основную (наиболее важную) функциональность системы. Данное тестирование применяется для того, чтобы удостовериться, что основные, наиболее важные функции системы работают как должно, без того, чтобы углубляться в детали.

Дымовое тестирование принято проводить вначале любого другого тестирования, как показатель того, что сборка готова к тестированию. Если же сборка не проходит дымовое тестирование, то её отправляют на доработку. Дымовое тестирование в этом контексте называют «проверочным испытанием сборки» (build verification test).

Необходимо помнить, что дымовое тестирование не «заглядывает» глубоко в систему, оно лишь проверяет основную функциональность сборки. Например, если приложение оконное, то тестировщик захочет проверить, открывается ли окно приложения вообще и будет ли управляющие элементы на окне в принципе приводить к исполнению некоторых действий.

Таким образом, цель дымового тестирования состоит в том, чтобы проверить – возможно, приложение настолько повреждено (нефункционально), что проводить более детальные тесты просто не имеет никакого смысла.

Так как дымовое тестирование не углубляется в структуру приложения, а концентрируется на его поведении и исследует только основную функциональность, то очевидно, что оно связано с малыми затратами временных и человеческих ресурсов, а значит может использоваться повсеместно. Поэтому создание ежедневных сборок приложения и проведение ежедневного дымового тестирования – это мировая практика, которая хорошо себя заявила.

GUI Testing

Тестирование пользовательского интерфейса (Graphics User Interface testing – GUI testing) – это важная часть любого процесса тестирования. Поскольку, пользовательский интерфейс – лицо приложения. Скрытая от глаз пользователя функциональность может быть реализована не идеально и не сверхпроизводительно, но этого не видно пользователю, тогда как недостатки интерфейса (нестабильность, сложность, незаконченность, недоверие пользователя к интерфейсу) могут заставить конечного пользователя отказаться от использования продукта.

Поскольку сложно формально оценить оформление с точки зрения функциональности, то тестирование графического пользовательского интерфейса сосредоточено на исследовании компоновки и реализации элементов управления.

Для формальной оценки функциональности пользовательского интерфейса вводиться понятие полноты, которое находи своё выражение в стандартах полноты пользовательского интерфейса. Полноту можно понимать как весь спектр функциональности, который предоставляет пользователю возможность осуществлять все необходимые управляющие воздействия, а также осуществлять доступ ко всем необходимым представлениям данных, получаемых в ходе работы приложения.

Поскольку наиболее распространённым GUI на сегодняшний день является оконный интерфейс, то говорят о полноте оконного интерфейса.

Ниже приведён список проверки пользовательского интерфейса, который позволяет удостовериться в его полноте; его также можно назвать стандартами полноты оконного интерфейса:

- полнота любого приложения:
  - приложение должно стартовать при двойном щелчке на иконке приложения;

- окно загрузки должно предоставлять пользователю такую информацию, как название, версия приложения, информацию о разработчике, иконку и т.д.;
- главное окно приложения должно иметь тот же заголовок, что и иконка в окне Program Manager (установка/удаление программ);
- закрытие приложения должно приводить к сообщению: «Вы уверены в том, что хотите закрыть программу?»;
- должно быть реализовано поведение, позволяющее запускать программу более чем в одном экземпляре, если обратное не предусмотрено архитектурой системы;
- на любом приложении, если приложение занято, курсор должен изменяться на часы или должен использоваться другой механизм, позволяющий уведомить пользователя о том, что приложение выполняет некоторые операции и не может отвечать на его запросы;
- обычно клавиша F1 используется для вызова справки. Если приложение использует некоторую интегрированную систему «помощи» (справки), то она должна инициироваться нажатием клавиши F1;
- функции минимизации и восстановления окна должны работать должным образом;

- полнота каждого окна приложения:
  - заголовок окна для каждого приложения должен содержать название приложения и название окна. Если произошла ошибка, то и сообщение о ней;
  - заголовки окон и информационные сообщения должны быть понятны пользователю;
  - если предусмотрены системные меню, такие как «переместить», «изменить размер», «закрыть», то необходимо их использовать;
  - тестовые представления должны быть проверены СС точки зрения синтаксиса и грамматики;

- если предусмотрена навигации посредством клавиши TAB, то нажатие этой клавиши должно перемещать фокус по элементам управления, в прямом порядке, а комбинация SHIFT+TAB – в обратном;
- перемещение фокуса посредством TAB должно быть организовано сверху вниз и слева направо;
- если элемент управления может принимать фокус, то это состояние должно отражаться наличием пунктирной линии вокруг элемента управления (или определённым стилем интерфейса способом);
- пользователь не должен иметь возможности выделять или другим образом взаимодействовать с неактивными элементами управления (как правило, выделенными серым цветом);
- текст должен быть выровнен по левому краю, а абзацы ещё и по ширине;
- как правило, все основные команды должны иметь возможность запуска посредством комбинации клавиш быстрого доступа;

- текстовые поля (text boxes):
  - если текстовое поле редактируемое, по перемещение фокуса на него должно приводить к переходу его в состояние редактирования (появлению курсора), а если не редактируемое, то оно должно оставаться неизменным;
  - необходимо протестировать текстовое поле на переполнение буфера, вводя в него столько символов, сколько это возможно;
  - необходимо протестировать текстовое поле на то, как оно обрабатывает специальные символы и не допустимые символы и проверить, что не возникает аномалий в поведении текстового поля;
  - Пользователь должен иметь возможность выделять текст в текстовом поле посредством комбинации клавиши Shift + нажатия клавиши со стрелкой. Выделение, также, должно быть возможно посредством

мыши, а двойное нажатие кнопки мыши должно выделять всё содержимое текстового поля;

- переключатели (radio buttons):
  - только одна из предлагаемых альтернативных опций должна быть выделена в один момент времени;
  - пользователь должен иметь возможность выбрать любую из кнопок группы опций посредством мыши или клавиатуры;
  - клавиши со стрелками должны изменять состояние переключателя (выделен/невыделен);
- выключатели или флажки (check boxes):
  - пользователь должен иметь возможность выбрать любую комбинацию включённых флажков;
  - нажатие кнопки мыши или пробела на флажке должно изменять его состояние (включён/выключен);
- кнопки (push buttons);
  - все кнопки, за исключением OK/Cancel, должны иметь клавиши быстрого доступа к ним. Эта возможность должна визуализироваться посредством подчёркнутой буквы в тексте кнопки. Кнопка также должна активироваться нажатием клавиши ALT;
  - нажатие мыши на любой кнопке должно активировать её и приводить к срабатыванию закреплённого за ней обработчика. Если на кнопке установлен фокус, то нажатие пробела и клавиши Enter должны приводить к тому же результату;
  - если на окне предусмотрена кнопка отмены, то нажатие клавиши Escape должно приводить к тому же результату;
- выпадающие меню (drop down list boxes):
  - нажатие стрелки на выпадающем списке должно приводить к появлению списка опций. Список должен поддерживать возможность прокрутки, но пользователь не должен иметь возможность вводить в него данные непосредственно;

- нажатие комбинации Ctrl+F4 должно приводить к разворачиванию списка;
- нажатие клавиши должно приводить к выделению в списке первого элемента, начинающегося с символа, соответствующего нажатой клавише (за исключением функциональных клавиш);
- элементы списка должны быть упорядочены по алфавиту;
- выделенный элемент должен отображаться в заголовке списка;
- должен быть только один пустой элемент в списке;

- комбинированные списки (Combo Box):
  - требования такие же, как и к выпадающим спискам, за тем исключением, что пользователь должен иметь возможность вводить в него данные;
- списки (List Boxes):
  - дожжен быть доступен одиночный выбор элемента списка как нажатием кнопки мыши, так и перемещением указателя посредством клавиатуры;
  - нажатие любой клавиши должно приводить к выделению первого элемента начинающегося с соответствующего символа;
  - если в интерфейсе имеются управляющие кнопки (вид/представление, открытие), то за ними должно быть закреплено соответствующее поведение;
  - Необходимо удостовериться в том, что все данные, представляемые списком можно просмотреть используя ползунок прокрутки.

Load testing

Нагрузочное тестирование (load testing) - это метод тестирования программного обеспечения, который состоит в том, чтобы проверить работоспособность системы в условиях нагрузок, превышающих штатные, а также пиковых нагрузок.

Понятие нагрузки происходит из физики и связано с условными ограничениями, которые накладываются на информационную систему средой исполнения (например аппаратными ограничениями); отсюда же происходит и понятие о минимальных и желательных конфигурациях платформы, на которой будет эксплуатироваться программный продукт.

Нагрузка описывается в терминах интенсивности, сложности и объёмности исполняемых операций (под операцией, в данном контексте, понимается функциональная единица программы; например, получение выборки из базы данных):

под *интенсивностью* понимается количество операций в единицу времени;

под *сложностью* – вычислительная сложность выполняемой операции (например, в L-нотации, или в количестве процессорных тактов, необходимых для выполнения операции);

объёмность определяется объёмом вычислений (количеством различных вычислений), необходимых для завершения операции (например, для расчёта растеризации трёхмерного изображения необходимо большое количество вычислений, имеющих большую вычислительную сложность; также примером может служить выборка большого количества данных из базы – вычисления сравнительно простые, однако количество вычислений объёмно).

Соответственно вышесказанному, нагрузочное тестирование заключается в создании нагрузки посредством комбинирования сложности, объёмности и интенсивности вычислений и фиксировании поведения системы в условиях различной нагрузки, которое должно удовлетворять заявленным спецификациям.

Performance Testing

Тестирование производительности (performance testing) – это набор методов тестирования, направленных на исследование производительности информационной системы. Производительность понимается как отношение объёма проделанной работы к времени, за которое она была выполнена. В контексте тестирования программного обеспечение предполагается, что тестирование производительности направлено на выяснение того, насколько быстро выполняются различные аспекты функционирования программного продукта при определённой нагрузке. Тестирование производительности находит выражение в пяти методах тестирования:

- нагрузочное тестирование (load testing);
- стрессовое тестирование (stress testing);
- тестирование надёжности (endurance testing) – тест производительности, направленный на определение или подтверждение характеристик производительности системы, при котором система подвергается определённым, ожидаемым, нагрузкам и обрабатывает определённый, ожидаемый, объём данных за некоторый (сравнительно большой) промежуток времени. Цель теста состоит в том, чтобы проверить, сможет ли система надёжно функционировать в ожидаемых условиях эксплуатации;
- тестирование пиковых нагрузок (spike testing) – тест производительности, который состоит в том, чтобы создавать тестируемой системе пиковые нагрузки на коротких промежутках времени в ожидаемом диапазоне нагрузок. Целью теста является изучение стабильности системы в ожидаемых условиях эксплуатации;
- тестирование мощности (capacity testing) – тест мощности, похож на стрессовое тестирование, однако, цель тестирования мощности отличается и состоит в том, чтобы выяснить максимально допустимые для системы нагрузки, максимальную нагрузку, при которой система перестанет функционировать.

Stress Testing

Стрессовое тестирование (stress testing) – это тест производительности, целью которого является изучение стабильности и, главным образом, устойчивости тестируемой системы.

Стрессовое тестирование состоит в том, чтобы изучить прочность, доступность и возможность обработки ошибок системы при высоких нагрузках. Другими словами, система всяческими способами выводиться из равновесия и изучается её поведение в заданных условиях. При нормальном функционировании, после выведения из равновесия система должна вернуться в первоначальное состояние – такое поведение называется устойчивостью системы.

Польза стрессового тестирования состоит в следующем:

- стрессовое тестирование позволяет определить могут ли данные быть повреждены в стрессовой ситуации;
- позволяет определить диапазон повышения нагрузки между штатной (ожидаемой) нагрузкой и моментом, в котором система прекратит выполнять основные функции или станет выполнять их недопустимым образом (например, слишком медленно, или некорректно);
- позволяет подтвердить, что в системе безопасности не появляются бреши в условиях перегрузок;
- позволяет определить побочные эффекты от провоцируемые аппаратным или поддерживаемым программным обеспечением.

Security Testing

Поскольку в сегодняшнем мире практически не возможно представить программного обеспечения, которое бы не содержало конфиденциальной информации: начиная с самого очевидного – web-приложений, поддерживающих пользовательские сеансы, а значит хранящих личную информацию пользователей; и заканчивая настольными приложениями. Казалось бы, что редактор изображений не содержат ничего

конфиденциального, однако на сегодняшний день даже ими поддерживают возможности аутентификации пользователя и получения по средствам коммуникации эксклюзивной информации (например, эксклюзивных плагинов, ресурсов и обучающего материала). Само собой, необходимо защитить канал передачи информации от несанкционированного проникновения, иначе лицензионные права собственника системы будут нарушены. И, естественно, излишне напоминать о необходимости защиты бизнес-информации, которая постоянно пересылается в распределённых системах бизнес-процессинга.

Подытоживая вышесказанное, хотим заметить, что там, где есть данные, необходимо принимать меры по защите от стороннего проникновения.

Таким образом, тестирование защищённости (security testing) – это вид тестирования информационных систем, целью которого является подтверждение (опровержение) того, что система защищает используемые ею данные, а также принимает меры по защите конфиденциальности пользователя, который её пользуется.

Тестирование защищённости начинается с классификации возможных уязвимостей, которые условно можно поделить на следующие основные группы:

- сканирование сети (network scanning);
- взлом паролей (password cracking);
- просмотр журналов программ (log review);
- взломщик целостности файла (file integrity cracker).

Разумеется, что тестировать системы необходимо в зависимости от того, каким функционалом она наделена. Если система не ожидает сетевых подключений, то сканирование сети ей не опасно. Или, если система не ведёт журналов (что, в принципе, вряд ли), то log review ей тоже неопасен.

Тестировщику не нужно знать, как можно защищаться от атак, его цель – установить, имеются ли уязвимости. Для этого необходимо знать пути проникновения в систему. С другой стороны знание о способах защиты упростит тестирование.

Наиболее распространённым видом атак являются атаки на пароли. Здесь мы приведём лишь некоторые сведения о важности сложности пароля, а также о том, какие пароли являются сложными для атак.

Обыватель редко понимает, что подразумевается под выражением «взлом пароля». Обычно в воображении пользователя возникает картина, так усердно навязываемая художественными фильмами, в которой некий «хакер», представляющийся таким себе феноменом, при помощи чудотворных процедур, до сих пор не понятным методом подбирает пароль. На самом деле подбор пароля – это самый неизобретательный способ получить доступ к информационной системе.

Большинство людей для формирования пароля используют личную информацию или из-за того, что не могут запомнить сложные пароли. Люди часто выбирают в качестве пароля личную информацию, поскольку пароль нужно придумать немедленно, и вспомнить незамедлительно. Поэтому, для удобства запоминания, многие используют имена своих близких, друзей, свои личные данные (такие как телефон, день рождения и др.). Если взломщик знаком с владельцем пароля, то может без труда подобрать пароль, основываясь на информации о его личности.

Также к этому виду атак относиться подбор, статистически наиболее часто используемых в качестве пароля, слов. Взломщиками составляются словари таких паролей, а потом специально программное обеспечение использует каждый из них, пока не подберёт нужный.

Также вирусное программное обеспечение может осуществлять автоматический поиск паролей, сохраненных другим программным обеспечением на пользовательском компьютере, в файлах пользователя или путём сканирования сигналов, поступающих с клавиатуры в те моменты, когда пользователь набирает пароль. Затем вирус отправляет эти данные злоумышленнику и он, имея точную информацию о имени пользователя и пароле, проникает в систему под видом этого пользователя.

Из представленной ниже таблицы видно, что и увеличение алфавита и увеличение размера слова влияют на надёжность приблизительно одинаково, а наибольшее число комбинаций даёт совместное увеличение и алфавита и разрядности пароля.

Поэтому необходимо, чтобы информационная система контролировала сложность пароля, придуманного пользователем, и предупреждала его о том, что он (пароль) либо очевиден, либо слабо защищён.

Связь между длиной и алфавитом пароля и количеством возможных комбинаций при подборе

Набор символов	**Длина пароля**				
	4-octet	**5-octet**	**6-octet**	**7-octet**	**8-octet**
Алфавитные символы в нижнем регистре (26)	$4.6x10^5$	$1.2x10^7$	$3.1x10^8$	$8.0x10^9$	$2.1x10^{11}$
Алфавитно-цифровые символы в нижнем регистре (36)	$1.7x10^6$	$6.0x10^7$	$2.2x10^9$	$7.8x10^{10}$	$2.8x10^{12}$
Алфавитные символы в обоих регистрах(62)	$1.5x10^7$	$9.2x10^8$	$5.7x10^{10}$	$3.5x10^{12}$	$2.2x10^{14}$
Печатные символы (95)	$8.1x10^7$	$7.7x10^9$	$7.4x10^{11}$	$7.0x10^{13}$	$6.6x10^{15}$
Семи-битные ASCII символы (128)	$2.7x10^8$	$3.4x10^{10}$	$4.4x10^{12}$	$5.6x10^{14}$	$7.2x10^{16}$
Восьми-битные ASCII символы (256)	$4.3x10^9$	$1.1x10^{12}$	$2.8x10^{14}$	$7.2x10^{16}$	$1.8x10^{19}$

Важным является вопрос выбора оптимальной длины пароля. Естественно, что чем больше число символов – тем лучше. Но при условии, что символы в слове расположены случайно, возникает проблема запоминания пароля владельцем. Из приведённой далее таблицы видно, что при размере алфавита в 256 возможных значений, и разрядности слова в 8 символов на процессоре, который может перебирать 1 миллион комбинаций в течении 1 секунды, для полного перебора всех возможных комбинаций потребуется 570,776 лет.

Можно бы было предположить, что 2283 года (для слова с разрядностью – 7 символов) – это вполне достаточно. Но необходимо помнить, что в реальной ситуации нам не придётся перебирать все комбинации, поскольку вероятность того, что искомая комбинация окажется последней в порядке подбора, - чрезвычайно мала. Также необходимо учитывать, что в реальности перебор редко осуществляется одним вычислительным средством – как правило, над задачей работает кластер (возможно бот-сеть). Поэтому подбор пароля из 7 символов представляется абсолютно реальным. Соответственно необходимо, чтобы пароль содержал минимум 8 символов.

Дальнейшее увеличение длины пароля не оправдано, поскольку резко усложняется его запоминание, учитывая, что символы расположены случайным образом.

Длину пароля при выборе его пользователем система тоже должна валидировать.

Время, необходимо для поиска всех возможных комбинаций (при условии, что осуществляется перебор 1 миллиона комбинаций в течении 1-ой секунды).

Набор символов	**Длина пароля**				
	4-octet	**5-octet**	**6-octet**	**7-octet**	**8-octet**
Алфавитные символы в нижнем регистре (26)	0.5 сек.	12 сек.	5.2 мин.	2.2 часов	2.4 дней
Алфавитно-цифровые символы в нижнем регистре (36)	1.7 сек.	1 мин.	36.7 мин.	21.7 часов	32.4 дней
Алфавитные символы в обоих регистрах(62)	15 сек.	15 мин.	15.8 часов	40.5 дней	7 лет
Печатные символы (95)	1.4 мин.	2.1 часов	8.6 дней	2.2 лет	209 лет
Семи-битные ASCII символы (128)	4.5 мин.	9.4 часов	50.9 дней	17.8 лет	2283 лет
Восьми-битные ASCII символы (256)	1.2 часов	12.7 дней	8.9 лет	2283 лет	570,776 лет

Installation Testing

Тестирование процесса установки или установочное тестирование (installation testing) – очень важный этап тестирования, поскольку, если будут существовать ошибки в процессе установки, то пользователь может не сумеет получить наш продукт и все усилия по тестированию и отладке системы будут напрасными.

Также установка – это первый опыт знакомства пользователя с нашим продуктом, поэтому необходимо, чтобы он проходил «гладко», чтобы у пользователя не возникало никаких проблем.

Процесс инсталляции зависит от многих факторов:

- значение имеет способ доставки пользователю программного обеспечения (важно, поставляется продукт на диске или же устанавливается удалённо, через сетевой инсталлятор, который скачивается пользователем с сайта производителя);
- второй важный момент – ориентирован ли наш продукт на одну платформу, или мы поддерживаем некоторое количество различных платформ.

Тестировщик, для того, чтобы начать тестирование, должен знать, чего следует ожидать от успешной установки, чтобы иметь возможность идентифицировать успешность и неудачу. В качестве критериев успешности можно брать такие показатели как количество файлов, наличие после установки дополнительных модулей (вспомогательных сервисов, ресурсов), наличие записей в реестре (если речь идёт об операционной системе Windows) и так далее.

Также необходимо убедиться в том, что инсталлятор предоставляет пользователю всю необходимую степень управляемости, чтобы обеспечить возможность настройки установки максимально близкой к потребностям пользователя (пользователь должен иметь возможность настроить приложение так, как ему необходимо).

Platform/Environment Testing

Тестирование взаимодействия с платформой/окружением (platform/environment testing) – виде тестирования, целью которого является установление того, что приложение нормально интегрируется в платформу (взаимодействует с платформой), под которой, как правило, понимается операционная система, или другую программную среду, в рамках которой она будет исполняться.

Достаточно важным является убедиться в том, что у разработанной системы отсутствуют конфликты с окружающей средой, а также, что окружающая среда не воспримет систему враждебно (например, не будет ли система блокирована антивирусным программным обеспечение) или конфликтовать с компонентами системы (например, в плане доступа к системным ресурсам).

Parallel/Comparison Testing

Параллельное тестирование или тестирование сравнением (parallel/comparison testing) – это подход к тестированию программного обеспечения, который заключается в поиске преимуществ и недостатков, слабых и сильных сторон в продукте, в сравнении с предыдущими версиями.

Информация, которая может быть получена в рамках данного исследования, позволяет составить представление о динамике развития продукта во времени (на протяжении выпуска новых версий), чтобы выявить возможные недостатки в постановке задачи на разработку, и корректировке стратегического плана развития разрабатываемого программного продукта.

Данное исследование является также полезным с точки зрения маркетинга, поскольку позволяет сравнить различные версии на предмет устаревания популярности некоторых компонент и сделать прогноз на дальнейшее развитие продукта.

Usability Testing

Тестирование удобства использования или тестирование юзабилити (usability testing) – это нефункциональный метод тестирования программного обеспечения, который направлен на изучение того, насколько пользовательский интерфейс системы удобен в использовании.

Удобство использования понятие субъективное, однако, возможно выделить и объективные характеристики, такие как, например, видимость элементов управления, читабельность текста, цветовая палитра, и прочие. Но, как бы то ни было, такое понятие как юзабилити не достаточно изучен (термин юзабилити заимствован из английского языка для краткости, и устоялся как общеупотребительное выражение, поскольку выражение «удобство использования» громоздко в употреблении).

Тестирование юзабилити может внести следующие преимущества:

- гораздо проще продавать продукт с высокой степенью юзабилити (high usable), поскольку, как показывает практика, рядовой пользователь будет отдавать предпочтение удобному продукту в сравнении с неудобным, даже если придётся пожертвовать функциональностью в допустимой степени;
- продукт с высокой степенью юзабилити легко изучать и использовать, поэтому он будет пользоваться более широким спросом;
- расходы на поддержку продукта с высокой степенью юзабилити меньше, нежели на аналогичный, с низкой степенью юзабилити, поскольку у пользователей возникает меньше проблем с настройкой и использованием инструментария.

Для тестирования удобства использования необходимо выделить метрики (формальные характеристики, которые можно измерить и формально оценить). Эти метрики могут изменяться в зависимости от характера и назначения системы. Однако существуют постоянные показатели, действительные для

любого проекта. Поскольку при тестировании юзабилити тестировщик имеет дело с интерфейсом, то можно выделить следующие объективные показатели юзабилити:

- прозрачность – пользователь должен понимать, какие именно процессы происходят в системе и какой результат будет получении при определённых действиях. Другими словами, если пользователь выбирает пункт меню «сохранить документ», то не должно происходить обновление приложения;
- дружелюбность – наверное, самая субъективная из характеристик интерфейса. Она предполагает, что интерфейс не должен действовать агрессивно по отношению к пользователю. То есть элементы управления не должны «скрываться» от пользователя в недоступных местах; не должно происходить блокировки интерфейса, без крайней необходимости. Пользователь должен ощущать удобство при использовании интерфейса;
- интуитивность – понятность интерфейса, которая выражается во времени, необходимом среднему пользователю, чтобы изучить интерфейс без дополнительной информации о том, как он устроен.

В тех случаях, когда выделение объективных метрик является невозможным, или же выделено крайне мало времени на испытание интерфейса на реальных пользователях, можно прибегнуть к методу экспертной оценки – пригласить эксперта, компетентного в области разработки интерфейса и попросить его оценить юзабилити по всем необходимым пунктам. Хотя предпочтительнее – проводить объективное оценивание.

Conformance Testing

Тестирование соответствия (conformance testing) используется для того, чтобы подтвердить, что разработанная система удовлетворяет требованиям стандартов соответствующих классу систем, к которому она относится.

Стандарты, разрабатываются в тех случаях, когда необходимо достигнуть высоких показателей эффективности и совместимости. Эффективность достигается за счёт того, что для формирования стандарта используется модель системы, которая хорошо себя зарекомендовала, и эффективность которой проверена временем. В данном контексте понятие эффективности совмещает в себе как функциональные качества системы, так и её структурные свойства.

Стандарт представляет собой набор спецификаций, которые условно сравнимы со спецификациями определёнными для системы. Поэтому, протестировать систему на соответствие стандартам можно также, как и на соответствие заявленным спецификациям, используя для этого все доступные подходы к тестированию.

User Acceptance/Alpha/Beta Testing

Приёмо-сдаточное тестирование (acceptance testing) – это формальное тестирование, предназначенное для определения того, удовлетворяет ли разработанное программное обеспечение потребности и ожидания пользователя и предоставляет конечному пользователю возможность определить: покупать ему систему или нет; также приёмо-сдаточное тестирование позволяет в реальных условиях эксплуатации проверить все функциональные и нефункциональные характеристики разработанной системы и получить от пользователей обратную связь, для внесения завершающих изменений и устранения выявленных дефектов.

Приёмо-сдаточное тестирование может принимать две нижеследующие формы:

- альфа-тестирование (alpha testing) – состоит в том, чтобы опротестировать системы на стороне производителя конечными пользователями и характеризуется тем, что разработчик может косвенно контролировать окружение системы;
- бета-тестирование (beta testing) – происходит на стороне пользователя и, соответственно, команда разработчиков не имеет никакого контроля над системой.

Начинать приёмо-сдаточное тестирование следует только после описанных ниже условий:

- системное тестирование завершено и дефекты, выявленные за время системного тестирования, устранены или были документированы;
- план приёмо-сдаточного тестирования готов;
- окружение для приёмо-сдаточного тестирования готово.

Условия завершения приёмо-сдаточного тестирования:

- приёмочное описание для разработанной системы сформировано;
- команда разработчиков уведомлена обо всех предостережениях и отчётах пользователей об выявленных ошибках.

Для приёмо-сдаточного тестирования достаточно важно сформировать с потенциальными покупателями подробные приёмочные критерии ещё на этапе разработки. С одной стороны, это поможет команде разработчиков создать систему, наиболее адекватно удовлетворяющую пользовательские потребности; а, с другой – станет проще определить окончание приёмо-сдаточного тестирования, вследствие наличия объективных критериев тестирования.

Другие типы тестирования

Как можно было заметить, существуют и такие виды тестирования, которые не были описаны в текущем уроке, однако, по ходу изложения о них упоминалось: тестирование масштабируемости (scalability testing), тестирование вменяемости (sanity testing), тестирование обработки приложением ошибок (error handling testing) и так далее. Существуют также подходы, о которых в данном уроке не было упомянуто. Например, автоматизированное тестирование (automated testing), которое скорее является не методом, а подходом. Автоматизированное тестирование лишь организует выполнение известных методик тестирования с использованием средств автоматизации (как правило, программных; но ведь автоматизацию можно рассматривать и в более широком смысле), позволяющих ускорить процесс тестирования, а значит сделать его более эффективным и более дешёвым.

Описать всё в рамках одного урока – невозможно, и мы вынуждены ограничиваться лишь некоторым спектром методов. Нами были рассмотрены наиболее популярные, на сегодняшний день, типы и подходы к тестированию.

Но, ещё раз напомним, что успешность тестирования состоит в том, чтобы адекватно задачам подобрать необходимые методики. Другими словами: применять методы там, где их использование целесообразно. Поэтому не стоит ограничиваться описанным инструментарием.

2. Цели и задачи документации в процессе тестирования

Как всякий формальный процесс, тестирование требует подробной документации; возможно, даже более подробной, чем при разработке.

Мы все знаем, что документацию вести необходимо, …. Но почем? Вот главный вопрос.

Дело в том, что тестирование процесс сложный, комплексный, состоящий из множества фаз и этапов. И все эти этапы взаимосвязаны. Например, нельзя начинать системное тестирование, пока не будут проведены модульное и интеграционное тестирование. А как мы можем быть уверены в том, что мы закончили некоторый этап, если мы не фиксируем свои действия. Или как мы можем быть уверены, что разработчики исправят ошибку, если мы не зафиксировали, что она была обнаружена. И даже, если разработчики узнают, что ошибка обнаружена в процессе тестирования, то без подробного описания последовательности действий, приводящей к возникновению ошибки, невозможно понять природу самой ошибки, а значит и исправить её.

Мы подошли ещё к одному свойству тестирования – тестирование предполагает взаимодействие между отделами, что приводит к дополнительным сложностям организации труда.

Собственно документация, в её широком понимании, и призвана решать организационные проблемы.

К целям документирования процесса тестирования относятся:

- организационная – состоит в том, что документация определяет очерёдность этапов и форму организации всего процесса тестирования. Например, тестовый план определяет порядок выполнения и характер проводимых тестов;
- управляющая – состоит в том, что документация определяет и разграничивает сферы ответственности исполнителей. Например, тестировщик, заполняющий отчёт об ошибке берёт та себя

ответственность за то, что описанная ошибка имела место и была обнаружена в указанное в отчёте время;

- систематизирующая – состоит в том, что документация систематизирует информацию, получаемую в процессе тестирования, а также фиксирует различные показатели процесса разработки программного обеспечения и состояние проекта в целом. Например, подобный эффект может выражаться в фиксации средствами документации всех обнаруженных дефектов, их общего количества, а также динамики появления на протяжении всего проекта. Также фиксация и систематизации информации о том, какие дефекты имеют место, может быть использована с целью применения мер, по недопущению возникновения таких дефектов в последующих проектах.

Но основная цель применения документации состоит в том, чтобы регламентировать процесс тестирования, организовать ход его выполнения, а также обеспечить его средствами фиксации всей информации, которая может понадобиться для обеспечения качества разрабатываемого программного продукта.

3. План тестирования (Test Plan)

Что такое план тестирования

Тестовый план или план тестирования – наиболее важный документ из всего спектра документов, сопровождающих процесс тестирования программного обеспечения. Он регламентирует все мероприятия, проводимые в рамках одной итерации процесса тестирования и содержит описание стратегии, которая будет применена для подтверждения того, что система отвечает выдвигаемым к ней требованиям.

Как правило, план тестирования должен содержать описание нижеследующих аспектов процесса тестирования:

- предмет тестирования – перед началом тестирования обязательно необходимо определить, какие стороны системы будут тестироваться, а какие не будут. Необходимо также знать, отвечает ли команда за проведение всего спектра тестирования по заданной предметной области или необходимо провести только некоторые определённые виды тестов;
- ссылки на сопутствующую документацию;
- управление рисками – необходимо описать, какие риски таит в себе тестируемая система (риски, связанные с не выявленными ошибками). Выбирая стратегию тестирования необходимо убедиться в том, что она покрывает все возможные риски;
- тестовое окружение – необходимо указать информацию о том, какое окружение будет использоваться для тестирования. Поскольку существует разница: тестировать системы в реальном окружении или в лабораторных (контролируемых) условиях;
- определение критериев – необходимо чётко определить критерии начала, приостановки, возобновления или полной остановки тестирования. Необходимо также чётко определить, когда можно считать, что тестирование окончено;
- график работ – как правило, начало тестирования зависит от расписания мероприятий, связанных с разработкой системы. Как только окончено формирование плана разработки, то можно приступать к составлению графика работ связанных с тестированием;
- ресурсы, необходимые для осуществления тестирования – в качестве ресурсов, имеются в виду аппаратное, программное обеспечение процесса тестирования, а также человеческие ресурсы, задействованные в процессе тестирования;

- используемые инструменты и средства автоматизации – необходимо указать информацию об инструментарии, который будет использоваться для тестирования (происходит из выбранной стратегии), поскольку эта информация может потребоваться как для организации процесса, так и для установления природы выявленных в процессе тестирования дефектов;
- мероприятия и отчётность – в данном разделе необходимо указать, какие мероприятия будут применены для реализации различных этапов тестирования, а также какого рода отч1тность они предусматривают.

Как правило, тестовый план составляется менеджером группы тестировщиков (test manager), главой отдела тестирования (test lead) или главным тестировщиком в группе (senior tester). Перед составлением плана тестирования должна быть собрана вся возможная информация о предмете тестирования, которая находит своё отражение в тестовом плане.

Планы тестирования имеют уровневую иерархию. Таким образом, вверху иерархии находиться мастер-план тестирования (master test plan), который определяет основную стратегию поведения и содержит все основные планы тестирования. Тогда как обычный план тестирования (test plan) детализирует организацию процесса конкретного этапа тестирования.

Цели и задачи плана тестирования

Можно выделить следующие цели создания плана тестирования:

- подготовка к тестированию, помогает продумать пути и способы тестирования;
- помочь другим участникам проекта, не находящимся в команде тестировщиков понять – что и каким образом тестировалось;
- создать средство регулирования и формализации процессов;

- создать документ, который бы описывал предметную область тестирования и используемые для тестирования стратегию и подходы;
- поддержка возможности обзора основных характеристик процесса тестирования со всей командой проекта.

Когда необходимо разрабатывать план тестирования?

Классически, план тестирования составляется тогда, когда уже реализован код, подлежащий тестированию. Но, разумеется, до того, как тестирование было начато. Такой подход к составлению тестовых планов имеет один изъян – неэффективный расход времени. Поскольку, за время когда составляется тестовый план, никаких работ не проводиться. Однако имеется преимущество – на момент составления тестового плана имеется вся необходимая информация о предмете тестирования.

Однако существует другой подход, который состоит в том, чтобы начать составление тестового плана после того, как были определены спецификации и ещё не была начата разработка. Преимущество данного подхода состоит в том, что при составлении тестового плана в самом начале жизненного цикла проекта, имеется возможность обнаружить ошибки и несоответствия в спецификациях ещё до начала реализации (кодирования). Недостаток же состоит в том, что тестовый план придётся изменить, если будут изменены спецификации (что на ранних этапах жизненного цикла проекта не редкость).

Шаблон плана тестирования в формате IEEE 829

Институт электроники и электронной инженерии (The Institute of Electrical and Electronics Engineers - IEEE) – это международная профессиональная организация, целью которой является развитие технологий

связанных с электроникой. Институтом IEEE был разработан стандарт известный как 829 – стандарт в области тестирования программного обеспечения. Этот стандарт определяет набор документов, использующихся в восьми различных стадиях процесса тестирования, каждый из которых потенциально должен сопровождаться отдельным видом документа. В рамках этого стандарта разработан шаблон плана тестирования, который включает все наиболее важные аспекты, которые необходимо отобразить в плане тестирования для любого вида случая тестирования.

Разделы плана тестирования

Test Plan Identifier, Authors and Revision History (идентификатор тестового плана, автор и история изменений): в данном разделе указывается уникальный идентификационный номер тестового плана, сгенерированный компанией-разработчиком, который уникально идентифицирует данный тестовый план, его уровень и уровень программного обеспечения, для которого он сформирован. Также в данном разделе необходимо указать сведения об авторе плана тестирования и истории изменений, внесённых в проект.

References (ссылки на сопровождающие документы)

Данный раздел должен содержать перечисление документов, сопровождающих план тестирования, с указанием их значения и причины прикрепления к плану.

Introduction (вступление)

Данный раздел содержит описание цели плана тестирования (постановку задачи), с возможным указанием уровня плана тестирования.

В данном разделе, также могут быть указаны ссылки на другие планы тестирования, документы или другие ресурсы, которые могут содержать информацию, относительно проекта. В случае, когда ссылки указаны в данном разделе, раздел references можно опустить.

Software Risk Issues (вопросы рисков)

В текущем разделе описываются риски, о которых говорилось выше.

Test Items (тестовые задания)

В настоящем разделе должны быть указаны тестовые задания (то, что должно быть протестировано). Тестовые задания могут быть разработаны на основе целей проекта и спецификаций. При формировании тестовых заданий могут быть использованы такие вспомогательные документы, как спецификации требований к проекту, пользовательские руководства, и другие.

Features to be Tested (качества, которые должны быть протестированы)

Данный раздел содержит перечисление качества, которые подлежат тестированию. Имеется в виду не техническое описание функциональности, а представление о них, с точки зрения пользователя («что система должна делать»). Настоятельно рекомендуется в этом разделе связать соответствующие этапы модели теста с соответствующими качествами.

Features not to be Tested (качества, не подлежащие тестированию)

Этот раздел содержит перечисление качеств системы, которые не нужно тестировать. Опять же качества рассматриваются не как технические спецификации некоторых функций системы, а их представление с точки зрения пользователя.

Approach (подход)

В этом разделе описывается используемый подход к тестированию – стратегия тестирования, которая используется в данном плане тестирования. Стратегия должна соответствовать уровню плана тестирования и быть приведена в соответствие со всеми выше и нижестоящими планами тестирования.

Должны быть указаны следующие основные процессы и правила:

- Будут ли использованы специальные инструменты.
  - Требуют ли используемые инструменты специальной подготовки.
- Какие метрики будут собираться в процессе тестирования.
  - На каком уровне будет собираться каждая метрика.
- Как будет обрабатываться процесс изменения конфигурации.

- Какое количество различных конфигураций будет протестировано.
  - Аппаратные.
  - Программные.
  - Комбинированные.
- Каковы правила регрессионного тестирования.
- Как будут обрабатываться требования и элементы модели, которые являются нестабильными или бессмысленными.
- Если это мастер-план тестирования, то необходимо укать общую стратегию и покрытие требований.
- Существуют ли какие-либо специфические требования к тестированию.
- Дополнительная информацию, которая может быть полезна при тестировании.
- Как буду проводиться собрания и другие организационные мероприятия.
- Существуют ли какие-либо специфические ограничения на тестирование.
- Есть ли какие-либо рекомендованные техники тестирования, которые должны быть использованы. Если да, то почему.

Item Pass/Fail Criteria (критерии выполнения тестовых заданий)

В данном разделе необходимо указать критерии, которые позволят определить, что тестовые задания были пройдены или «провалены». Критерии должны быть указаны соответствии с уровнем плана тестирования.

Suspension Criteria and Resumption Requirements (критерии приостановки и условия возобновления)

Необходимо указать критерии приостановки тестирования и условия, при которых тестирование, в соответствии с текущим планом тестирования, можно будет возобновить.

Test Deliverables (конечные результаты испытаний)

В данном разделе необходимо указать спектр отчётности, которая будет оформляться как результат тестирования. Это могут быть:

- план тестирования;
- спецификации модели теста;
- спецификации тестовых ситуаций;
- спецификации тестовых процедур;
- журнал (лог) теста;
- отчёт об обнаруженном дефекте;
- полный отчёт о тестировании.

Environmental Needs (потребности среды окружения)

Необходимо указать существуют ли какие-либо специальные требования к конфигурации окружающей среды для выполнения тестирования. Например:

- специальное аппаратное обеспечение;
- специальные требования к питанию;
- специфические версии вспомогательного программного обеспечения;
- специальные инструменты.

Staffing and Training Needs (необходимость специальной подготовки и другое)

В данном разделе необходимо указать – требуется ли для проведения тестирования специальная подготовка или любые другие средства, не указанные в предыдущих разделах.

Responsibilities (сферы ответственности)

Данный раздел содержит указание ответственных лиц для каждого мероприятия в рамках плана тестирования.

Schedule (график)

Данный раздел содержит описание временных рамок для всех этапов и мероприятия предполагаемых даням планом тестирования. Он должен составляться крайне аккуратно, основываясь на реальных датах. Если же, по каким-то причинам, разработка задерживается, то график приостанавливается вместе со всеми предполагаемыми им мероприятиями.

Planning Risks and Contingencies (планирование рисков)

Здесь должны быть указаны все риски, связанные с процессом тестирования. Примерами могут быть:

- запаздывание с доставкой аппаратного, программного обеспечения или инструментов;
- задержки во времени, связанные с обучением работы с приложением или инструментом;
- изменения в исходных требованиях или модели системы.

Approvals (подтверждение окончания тестирования)

В данном разделе определяется - кто отвечает за принятие решения об окончании процесса тестирования.

Glossary (глоссарий или список условных обозначений и сокращений).

4. Test Design

Что такое Test Design?

Моделирование теста (test design) – это процесс планирования и организации действий, необходимых для проведения теста. Моделирование теста процесс сложный, который начинается с создания плана тестирования и включает в себя все мероприятия по планированию в контексте тестирования.

Поскольку тестирование – это формальный процесс, то его проведение требует формализма – организации теста в виде набора документов, строго регламентирующих действия тестировщика.

Формализация начинается с того, что определяется список тестируемых качеств программного продукта, и заканчивая тем, что указывается конечное число возможных состояний системы, подлежащих тестированию.

Виды документов для тестирования

Checklist (список для проверки)

Одни из набора документов, сопровождающих процесс тестирования, цель которого состоит в том, чтобы, с одной стороны, смоделировать процесс тестирования путём спецификации качеств системы, подлежащих проверке, а, с другой – скомпенсировать потенциальные ограничения человеческих внимания и памяти. Суть такого документа. В его простейшей реализации, состоит в создании списка «что нужно сделать» (или списка проверки – «сто нужно проверить»), чтобы обеспечить максимальное покрытие теста. А в более расширенной форме представляет собой график, с указанием точного времени и формального определения действий, которые должны быть выполнены для проведения проверки.

Test Case (тестовая ситуация, тестовый случай)

Чёткий набор определённых характеристик систем, который идентифицирует её конкретное состояние, при котором тестировщик собирается определять, работает ли система надлежащим образом.

С формальной точки зрения тестовая ситуация – это документ, который идентифицирует состояние системы, в котором тестировщик собирается проверить соответствие системы заданному требованию (конкретной спецификации), а также описание того, какие действия будут выполнены для осуществления проверки. Должно быть как минимум два тестовых случая на каждое тестируемое требование: один для позитивного теста, а второй – для негативного.

Условно тестовый случай можно назвать просто тестом – он является наименьшей неделимой единицей всего процесса тестирования.

Test Scenario (тестовый сценарий)

Тестовый сценарий – это абстракция над тестовыми случаями. Он представляет собой строго определённую последовательность тестовых случаев (её можно представить в виде скрипта), которая предназначена для того, чтобы проверить, работает ли бизнес процесс или любой другой сложный

процесс, не сводимый к конкретному состоянию системы, от начала и до конца так, как это предполагается заданными спецификациями.

Test Suite/Test Design Specification/Test spec (набор тестов)

Набор тестов – это, как и тестовый сценарий, последовательность тестовых случаев, но преследующая другую цель. Цель набора тестов состоит в том, чтобы подтвердить, что система имеет специфический набор вариантов поведения.

Как правило, набор тестов содержит детальное описание инструкций или целей применения, которые указываются для каждой отдельной коллекции тестовых случаев, а также информацию о конфигурации системы, которая будет использована на протяжении тестирования.

Иногда наборы тестов просто используются для того, чтобы сгруппировать тесты вместе. Например для дымового тестирования может быть создан набор тестов, содержащий только дымовые тесты, и таким образом группирующий их в некоторый модуль.

Примеры создания документов

В качестве примера хотим привести checklist, составленный для тестирования метода генерации псевдослучайных чисел:

- значения, возвращаемые из метода, должны быть в рамках установленного диапазона;
- значения, возвращаемые из метода, должны быть равномерно распределены вдоль установленного диапазона;
- метод должен поддерживать возможность передачи ему в качестве аргументов значений всех числовых типов;
- метод должен возвращать результат более чем через секунду.

Другой пример построения checklist-а уже приводился нами в главе, посвящённой тестированию интерфейса.

5. Отчеты об ошибках (Bug Reporting)

Что такое отчет об ошибках?

Отчёт об ошибке – это сопровождающий процесс тестирования документ, назначение которого состоит в том, чтобы зафиксировать найденный тестировщиком дефект и отправить его в отдел разработки для исправления.

Жизненный цикл отчета

После обнаружения ошибки создаётся отчёт о найденном дефекте и закрепляется за разработчиком, который будет решать возникшую проблему, и ему присваивается состояние “assigned” (закреплён). После того, как проблема была решена разработчиком, багу присваивается состояние “resolved” (решён), и QA инженер или тестировщик делают повторную попытку обнаружения бага. Если баг не подтверждается, то ему присваивается состояние “closed” (закры) и отчёт с соответствующей записью отправляется в архив. Если же баг снова обнаруживается, то ему присваивается состояние “reopened” (заново открыт) и он отчёт снова отправляется в отдел разработки.

Бывают ситуации, когда дефект был обнаружен, отчёт о нём был сформирован, но после этого он ни разу не подтверждался. В таком случае багу присваивается состояние “unconfirmed” (не подтверждён), но отчёт всё равно отправляется в отдел разработки на исправление и проходит те же этапы. Которые были описаны выше.

Формат отчета

В каждой компании имеется свой собственный формат отчёта об ошибках, который содержит специфичные для принятого в компании процесса разработки программного обеспечения. В качестве универсальных полей отчёта об ошибке можно выделить следующие:

- идентификационный номер дефекта;
- описание;
- идентификационный номер версии сборки тестируемой системы;
- качество/модуль системы, в котором был обнаружен дефект;
- заголовок тестовой ситуации, в которой появился дефект;
- статус дефекта;
- приоритет;
- имя тестировщика;
- дата обнаружения дефекта;
- имя разработчика, за которым закреплён дефект;
- решение обнаруженной проблемы.

Примеры отчетов об ошибках

ID:	13546851
Описание:	Возникает ошибка времени исполнения, при введении в поле «пароль» специальных символов.
Build ID:	0.0.12365
Модуль/качество:	Окно авторизации пользователя.
Заголовок	
Тестовая ситуация:	Тестирование GUI. Окно авторизации пользователя.
Статус:	открыт
Приоритет:	высокий
Обнаружено:	Ивановым Иваном Ивановичем
Дата обнаружения:	01/09/1998
Закреплено за:	Сидоровым Петром Васильевичем
Решение:	----

Системы отслеживания ошибок (bugtracking systems).

Система отслеживания ошибок/багов (bugtracking system) - это программное средство, которое позволяет QA инженерам и разработчикам быстро отслеживать ошибки, которые обнаруживаются в их работе, и оперативно на них реагировать. Также она автоматизирует документооборот, который связан с поиском и исправление дефектов программного обеспечения.

На сегодняшний день существует большое количество open source баг-треккеров, которые хорошо себя заявили с точки зрения функциональности и масштабируемости. А использование баг-трекктера считается полезным с точки зрения организации процесса разработки программного обеспечения.

6. Домашнее задание

В качестве домашнего задания Вам предлагается протестировать любое из когда-либо созданных Вами приложений, с обеспечением планирования и составлением всей сопутствующей документации.

Необходимо:

- чтобы тестируемое приложение использовало графический интерфейс пользователя;
- чтобы тестируемое приложение оперировало данными (подключение к базе данных или обработка файлов с данными).

Как минимум необходимо провести следующие виды тестирования:

- модульное тестирование;
- интеграционное тестирование;
- тестирование графического интерфейса пользователя;
- нагрузочное тестирование;
- объёмное тестирование.