Формат <кода>

Тестирование ПО

formatkoda.ru
2022

Буров Сергей

Качество программного обеспечения

Характеристики качества ПО:

- **Функциональность (Functionality)**
- **Надежность (Reliability)**
- **Удобство использования (Usability)**
- **Эффективность (Efficiency)**
- **Удобство сопровождения (Maintainability)**
- **Портативность (Portability)**

Формат <кода>

Тестирование, QC, QA

- **Тестирование ПО** (**Testing**) = процесс исследования/испытания ПО, имеющий своей целью проверку соответствия между реальным и ожидаемым поведением ПО на конечном наборе тестов, выбранных определённым образом (ISO/IEC TR 19759:2005).
- **Контроль качества (QC, Quality Control)**
- **Обеспечение качества (QA, Quality Assurance)**

Формат <кода>

Жизненный цикл тестирования

Стадия 1 - общее планирование и анализ требований

Стадия 2 - уточнение критериев приёмки

Стадия 3 - уточнение стратегии тестирования

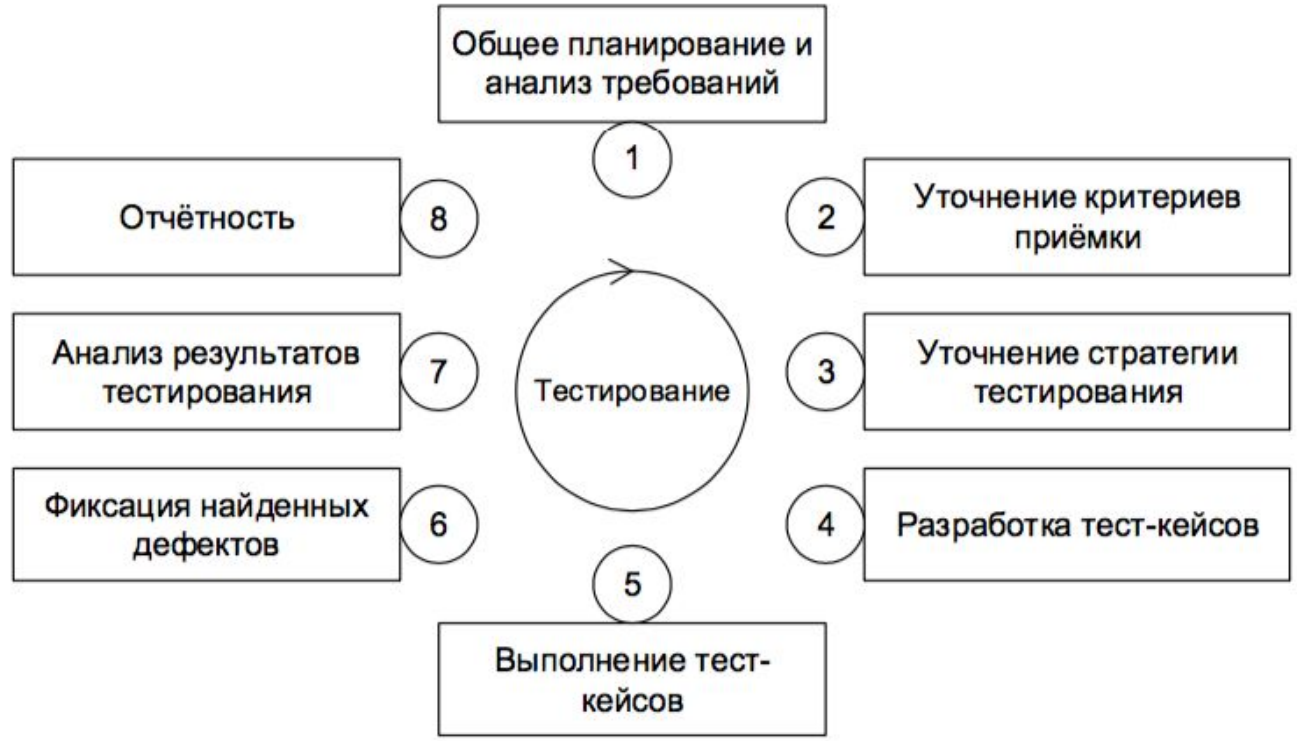
Стадия 4 - разработка тест-кейсов

Стадия 5 - выполнение тест-кейсов

стадия 6 - фиксация найденных дефектов

Стадия 7 - анализ результатов тестирования

стадия 8 - отчётность



Цели тестирования

- Проверка, все ли указанные **требования выполнены**
- Создание **уверенности в уровне качества** объекта тестирования.
- **Предотвращение дефектов**
- **Обнаружение** отказов и дефектов
- **Предоставление заинтересованным лицам достаточной информации**
- **Снижение уровня риска**

Принципы тестирования

1. Тестирование показывает наличие дефектов
2. Исчерпывающее тестирование невозможно
3. Раннее тестирование
4. Скопление дефектов
5. Парадокс пестицида
6. Тестирование зависит от контекста
7. Заблуждение об отсутствии ошибок.

Ошибка, дефект, сбой

- **Ошибка (error)** – это действие человека, которое порождает неправильный результат.
- **Дефект, Баг (Defect, Bug)** – недостаток компонента или системы, который может привести к отказу определенной функциональности.
- **Сбой (failure)** – несоответствие фактического результата (actual result) работы компонента или системы ожидаемому результату (expected result).

Причины возникновения ошибок

1. Недостаток или отсутствие общения в команде
2. Сложность программного обеспечения
3. Изменения требований
4. Плохо документированный код
5. Средства разработки ПО.

Артефакты тестирования

- План тестирования (Test plan)
- Чек-лист (check list)
- Тестовый сценарий (Test-case)
- Наборы тестовых сценариев (Test script or Test suite)
- Описание дефектов (Bug Report)
- Отчет о тестировании (Test-report)

Тестовый план

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 должен как минимум описывать следующее:

- **Что надо тестировать?**
- **Что будете тестировать?**
- **Как будете тестировать?**
- **Когда будете тестировать?**
- **Критерии начала тестирования**:
- **Критерии окончания тестирования**:
- **Какие РИСКИ возможны?**

Формат <кода>

Чек-лист

Чек-лист (check list) — это документ, который описывает что должно быть протестировано. Чек-лист может быть абсолютно разного уровня детализации.

Чек-лист должен обладать рядом важных свойств:

- Логичность
- Последовательность и структурированность
- Полнота и неизбыточность

Правила составления чек-листов:

- Одна операция
- Пункты чек-листа - это минимальные полные операции
- Пункты пишутся в утвердительной форме

Тест кейс

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

Атрибуты тест-кейса:

- **Уникальный идентификатор тест-кейса**
- **Название**
- **Предусловия**
- **Шаги**
- **Ожидаемый результат**
- **Постусловия**
- **Приоритет**
- **Приложения**

Формат <кода>

Характеристики хорошего тест-кейса

- Независимость.
- Четкие формулировки
- Наличие детальной, но не избыточной информации
- Легкая диагностика ошибок
- Исследование соответствующей («Ту, которую надо») области приложения
- Не выполняет ненужных действий
- Воспроизводимость

Наборы тестовых сценариев

Наборы тестовых сценариев (Test script or Test suite) - совокупность тест-кейсов, выбранных с некоторой общей целью или по некоторому общему признаку.

В общем случае наборы тест-кейсов можно разделить на:

- Свободные
- Последовательные

Описание дефектов

- **Дефект (bug)** — отклонение фактического результата от ожидаемого.
- **Ожидаемый результат** — поведение системы, описанное в требованиях.
- **Фактический результат** — поведение системы, наблюдаемое в процессе тестирования.
- **Отчёт о дефекте (bug report)** — это документ, описывающий ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

Атрибуты отчета о дефекте

- Уникальный идентификатор (ID)
- Имя (Тема, краткое описание, Summary)
- Подробное описание (Description)
- Шаги для воспроизведения (Steps To Reproduce)
- Фактический результат (Actual result)
- Ожидаемый результат (Expected result)
- Вложения (Attachments)
- Серьёзность дефекта (важность, Severity)
- Приоритет дефекта (срочность, Priority)
- Статус (Status)
- Окружение (Environment)

Severity vs Priority

Серьёзность (severity) показывает степень ущерба, который наносится проекту существованием дефекта

Градация Серьезности дефекта (Severity):

- **Блокирующий (S1 – Blocker)**
- **Критический (S2 – Critical)**
- **Значительный (S3 – Major)**
- **Незначительный (S4 – Minor)**
- **Тривиальный (S5 – Trivial)**

Срочность (priority) показывает, как быстро дефект должен быть устранён

Градация Приоритета дефекта (Priority):

- **P1 Высокий (High)**
- **P2 Средний (Medium)**
- **P3 Низкий (Low)**

Формат <кода>

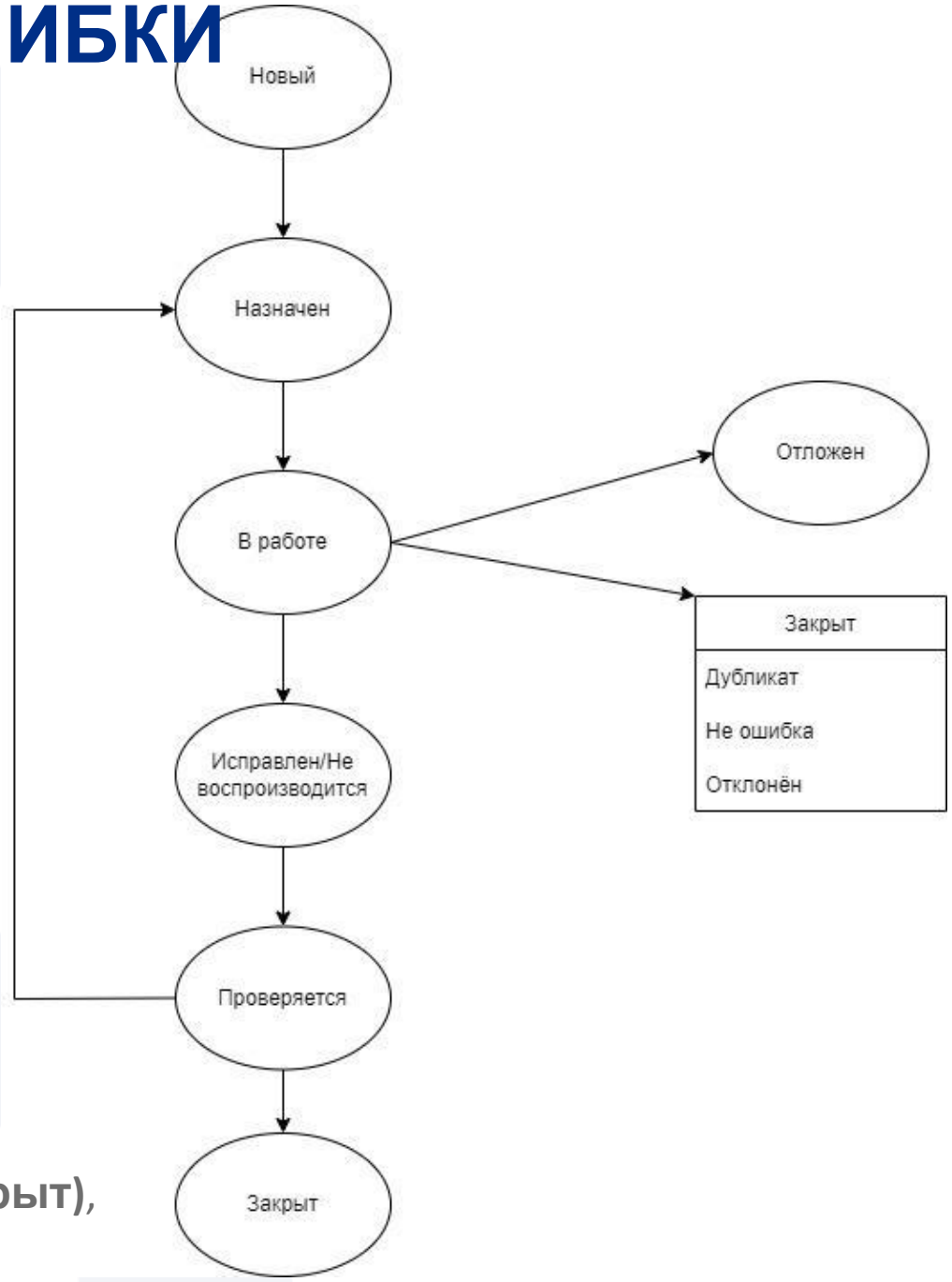
Логика создания эффективных отчётов о дефектах

1. Обнаружить дефект.
2. Понять суть проблемы.
3. Воспроизвести дефект.
4. Проверить наличие описания найденного вами дефекта в системе управления дефектами.
5. Сформулировать суть проблемы в виде «что сделали, что получили, что ожидали получить».
6. Заполнить поля отчёта, начиная с подробного описания.
7. После заполнения всех полей внимательно перечитать отчёт, исправить неточности и добавить подробности.
8. Ещё раз перечитать отчёт, т.к. в пункте 6 вы точно что-то упустили.

Формат <кода>

СТАДИИ ЖИЗНЕННОГО ЦИКЛА ОШИБКИ

1. Тестировщик обнаруживает дефект.
2. Тестировщик пишет отчет об ошибке (статус **New (новый)**)
3. Дефект назначен (статус **Assigned (назначен)**).
4. Разработчик изучает ошибку (В работе) и по полученным результатам соотносит ее к одному из статусов:
    1. **Duplicate (дубликат)**
    2. **Rejected (отклонен)**
    3. **Deferred (отсрочен)**
    4. **Not a bug (не баг)**
    5. **Fixed (исправлен)**
5. Тестировщик повторно проверяет ошибку (статус «**Retesting» (повторное тестирование)**).
6. Если дефект исправлен, тестировщик его закрывает (статусы «**Verified» (проверен),** «**Closed» (закрыт)**).
7. Если дефект проявляется и дальше, он опять передается на редактирование разработчику (статусы «**Reopened» (переоткрыт)**, «**Assigned» (назначен))**



Отчет о тестировании

Отчёт о результатах тестирования - документ, обобщающий результаты работ по тестированию и содержащий информацию, достаточную для соотнесения текущей ситуации с тест-планом и принятия необходимых управленческих решений.

Отчёт о результатах тестирования включает следующие разделы:

- **Краткое описание**
- **Команда тестировщиков**
- **Описание процесса тестирования**
- **Расписание**
- **Статистика по новым дефектамСписок**
- **Статистика по всем дефектам**
- **Рекомендации**
- **Приложения**

Формат <кода>

Типы (Виды) тестирования

Типы(Виды) тестирования

1. **Классификация по запуску кода на исполнение:**
   - **Статическое тестирование**
   - **Динамическое тестирование**
2. **Классификация по доступу к коду и архитектуре (Знанию системы):**
   - **Тестирование белого ящика** (white box)
   - **Тестирование чёрного ящика** (black box)
   - **Тестирование серого ящика** (grey box)
3. **Классификация по уровню детализации приложения:**
   - **Компонентное (модульное) тестирование(component/unit testing)**
   - **Интеграционное тестирование (integration testing)**
   - **Системное тестирование (system/end-to-end testing)**
   - **Приёмочное тестирование**

Формат <кода>

Типы(Виды) тестирования

4. **Классификация по степени автоматизации:**
   - Ручное тестирование (manual testing)
   - Автоматизированное тестирование (automated testing)
   - Полуавтоматизированное тестирование (semiautomated testing)
5. **Классификация по принципам работы с приложением**
   - **Позитивное тестирование**
   - **Негативное тестирование**
6. **Виды тестирования в зависимости от подготовленности:**
   - **Интуитивное тестирование**
   - **Исследовательское тестирование**
   - **Тестирование по документации**

Типы(Виды) тестирования

7. Связанные с изменениями виды тестирования:
   - Подтверждающее тестирование (Re-testing)
   - Дымовое тестирование (smoke test)
   - Санитарное тестирование (Sanity Testing)
   - Регрессионное тестирование (regression testing)
   - Тестирование сборки (Build Verification Test)
8. Классификация в зависимости от времени проведения:
   - Альфа-тестирование
   - Бета-тестирование
   - Гамма-тестирование(gammatesting)

Типы(Виды) тестирования

9. Классификация в зависимости от целей тестирования:

- **Функциональные виды:**
  - **Функциональное тестирование (functional testing)** — направлено на проверку корректности работы функциональности приложения.
  - **Тестирование безопасности (Safety Testing)** – тестирование программного продукта с целью определить его способность при использовании оговоренным образом оставаться в рамках приемлемого риска причинения вреда здоровью, бизнесу, программам, собственности или окружающей среде.
  - **Тестирование защищенности (Security Testing)** – тестирование с целью оценить защищенность программного продукта от внешних воздействий (от проникновений).На практике зачастую под термином тестирование безопасности понимают в том числе и тестирование защищенности.
  - **Тестирование взаимодействия (Interoperability Testing)** – это функциональное тестирование, проверяющее способность приложения взаимодействовать с одним и более компонентами или системами

Типы(Виды) тестирования

9. Классификация в зависимости от целей тестирования:
   - Нефункциональное тестирование (non-functional testing):
     - Тестирование производительности (performance testing)
     - тестирование Установки (installation testing)
     - Тестирование интерфейса (GUI/UI testing)
     - Тестирование удобства использования (usability testing)
     - Конфигурационное тестирование (Configuration Testing)
     - Тестирование на отказ и восстановление (Failover and Recovery Testing)
     - Тестирование локализации (localization testing)

Классификация по запуску кода на исполнение

- **Статическое тестирование (static testing)** —тестирование без запуска кода на исполнение. При этом, само тестирование может быть как ручным, так и автоматизированным.

В рамках этого подхода тестированию могут подвергаться:

- Документы (требования, тест-кейсы, описания архитектуры приложения, схемы баз данных и т. д.).
- Графические прототипы (например, эскизы пользовательского интерфейса).
- Код приложения (что часто выполняется самими программистами в рамках аудита кода (code review), являющегося специфической вариацией взаимного просмотра в применении к исходному коду).
- Параметры (настройки) среды исполнения приложения.
- Подготовленные тестовые данные.

- **Динамическое тестирование (dynamic testing)** —тестирование с запуском кода на исполнение

Запускаться на исполнение может:

- код всего приложения целиком (системное тестирование)

Формат <кода>

Классификация по доступу к коду и архитектуре (Знанию системы

- **Тестирование белого ящика** (white box) — у тестировщика есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного.

Преимущества:

- тестирование может производиться на ранних этапах: нет необходимости ждать создания пользовательского интерфейса;
- можно провести более тщательное тестирование с покрытием большого количества путей выполнения программы.

Недостатки:

- для выполнения тестирования белого ящика необходимо большое количество специальных знаний;
- при использовании автоматизации тестирования на этом уровне поддержка тестовых скриптов может оказаться достаточно накладной, если программа часто изменяется.

Классификация по доступу к коду и архитектуре (Знанию системы

- **Тестирование чёрного ящика** (black box)— метод тестирования ПО, который не предполагает доступа (полного или частичного) к системе. Основывается на работе исключительно с внешним интерфейсом тестируемой системы.

Преимущества:

- тестирование производится с позиции конечного пользователя и может помочь обнаружить неточности и противоречия в спецификации;
- тестировщику нет необходимости знать языки программирования и углубляться в особенности реализации программы;
- тестирование может производиться специалистами, независимыми от отдела разработки, что помогает избежать предвзятого отношения;
- можно начинать писать тест-кейсы, как только готова спецификация.

Недостатки:

- тестируется только очень ограниченное количество путей выполнения программы;
- без четкой спецификации (а это скорее реальность на многих проектах) достаточно трудно составить эффективные тест-кейсы;
- некоторые тесты могут оказаться избыточными, если они уже были проведены разработчиком на уровне модульного тестирования.

Классификация по доступу к коду и архитектуре (Знанию системы)

- **Тестирование серого ящика** (grey box) — метод тестирования ПО, который предполагает частичный доступ к коду проекта (комбинация White Box и Black Box методов).

Классификация по уровню детализации приложения

- **Компонентное (модульное) тестирование(component/unit testing)** — проводится для тестирования какого-либо одного логически выделенного и изолированного элемента (модуля) системы в коде. Проводится самими разработчиками, так как предполагает полный доступ к коду.
- **Интеграционное тестирование (integration testing)** — предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами).
- **Системное тестирование (system/end-to-end testing)** — процесс тестирования системы, на котором проводится не только функциональное тестирование, но и оценка характеристик качества системы — ее устойчивости, надежности, безопасности и производительности.
- **Приемочное тестирование или приемо-сдаточное испытание (Acceptance Testing)** - формальный процесс тестирования, который проверяет соответствие системы требованиям

Классификация по уровню детализации приложения

- **Интеграционное тестирование (integration testing)** — 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)**
- **Сверху вниз (Top Down Integration)**
- **Большой взрыв ("Big Bang" Integration)**:

Формат <кода>

Классификация по уровню детализации приложения

- **Системное тестирование (system/end-to-end testing)** — процесс тестирования системы, на котором проводится не только функциональное тестирование, но и оценка характеристик качества системы — ее устойчивости, надежности, безопасности и производительности.

Можно выделить 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 (test cases), которые должны быть протестированы.

Классификация по уровню детализации приложения

- **Приемочное тестирование или приемо-сдаточное испытание (Acceptance Testing)** - формальный процесс тестирования, который проверяет соответствие системы требованиям

Проводится с целью:

- определения удовлетворения системой приемочным критериям;
- вынесения решения заказчиком или другим уполномоченным лицом принятия приложения.

Классификация по степени автоматизации

- **Ручное тестирование (manual testing)** - тестирование, в котором тест-кейсы выполняются человеком вручную без использования средств автоматизации
- **Автоматизированное тестирование (automated testing)** - предполагает использование специального программного обеспечения (помимо тестируемого) для контроля выполнения тестов и сравнения ожидаемого фактического результата работы программы.

  Существует несколько основных видов автоматизированного тестирования:

  - **Автоматизация тестирования кода**
  - **Автоматизация тестирования графического пользовательского интерфейса**
  - **Автоматизация тестирования API**
- **Полуавтоматизированное тестирование (semiautomated testing**

Преимущества	Недостатки
• **Скорость выполнения тест-кейсов может вразы и на порядки превосходить возможности человека.** • **Отсутствие влияния человеческого фактора в процессе выполнения тест-кейсов (усталости, невнимательности и т.д.)** • **Минимизация затрат при многократном выполнении тест-кейсов (участие человека здесь требуется лишь эпизодически).** • **Способность средств автоматизации выполнить тест-кейсы, в принципе непосильные для человека в силу своей сложности, скорости или иных факторов.** • **Способность средств автоматизации собирать, сохранять, анализировать, агрегировать и представлять в удобной для восприятия человеком форме колоссальные объёмы данных.** • **Способность средств автоматизации выполнять низкоуровневые действия с приложением, операционной системой, каналами передачи данных и т.д**	• Необходим высококвалифицированный персонал в силу того факта, что автоматизация —это «проект внутри проекта» (со своими требованиями, планами, кодом и т.д.) • Высокие затраты на сложные средства автоматизации, разработку и сопровождение кода тест-кейсов. • Автоматизация требует более тщательного планирования и управления рисками, т.к. в противном случае проекту может быть нанесён серьёзный ущерб. • Средств автоматизации крайне много, что усложняет проблему выбора того или иного средства и может повлечь за собой финансовые затраты (и риски), необходимость обучения персонала (или поиска специалистов). • В случае ощутимого изменения требований, смены технологического домена, переработки интерфейсов (как пользовательских, так и программных) многие тест-кейсы становятся безнадёжно устаревшими и требуют создания заново.

Классификация по принципам работы с приложением

- **Позитивное тестирование** — тестирование, при котором используются только корректные данные. Все действия с приложением выполняются строго по инструкции без никаких недопустимых действий, некорректных данных и т.д.
- **Негативное тестирование** — в работе с приложением выполняются (некорректные) операции и используются данные, потенциально при-водящие к ошибкам (классика жанра —деление на ноль)

Классификация в зависимости от подготовленности

- **Тестирование по документации (testcase based testing)** – формализованный подход, в котором тестирование производится на основе заранее подготовленных тест-кейсов, наборов тест-кейсов и иной документации.
- **Исследовательское тестирование** – частично формализованный подход, в рамках которого тестировщик выполняет работу с приложением по выбранному сценарию, который, в свою очередь, дорабатывается в процессе выполнения с целью более полного исследования приложения.
- **Свободное (интуитивное) тестирование(ad hoc testing)** — полностью не-формализованный подход, в котором не предполагается использования ни тест-кейсов, ни чек-листов, ни сценариев.

Связанные с изменениями виды тестирования

- **Подтверждающее тестирование (Re-testing) -** Подтверждающее тестирование направлено на проверку исправления бага.
- **Регрессионное тестирование (regression testing)** — тестирование уже проверенной ранее функциональности после внесения изменений в код приложения, для уверенности в том, что эти изменения не внесли ошибки в областях, которые не подверглись изменениям.
- **Дымовое тестирование (smoke test)** — тестирование, выполняемое на новой сборке, с целью подтверждения того, что программное обеспечение стартует и выполняет основные для бизнеса функции.
- **Санитарное тестирование (Sanity Testing) -** это узконаправленное тестирование, достаточное для доказательства того, что конкретная функция работает согласно заявленным в спецификации требованиям.
- **Тестирование сборки (Build Verification Test) -** Направлено на определение соответствия выпущенной версии критериям качества для начала тестирования.

Классификация в зависимости от времени проведения

- **Альфа-тестирование(alpha testing)** выполняется внутри организации-разработчика с возможным частичным привлечением конечных пользователей.
- **Бета-тестирование(betat esting)** выполняется вне организации-разработчика с активным привлечением конечных пользователей/заказчиков
- **Гамма-тестирование(gammatesting)** — финальная стадия тестирования перед выпуском продукта, направленная на исправление незначительных дефектов, обнаруженных в бета-тестировании.

Функциональные виды тестирования

- **Функциональное тестирование (Functional Testing)** – тестирование, основанное на сравнительном анализе спецификации и функциональности компонента или системы.
- **Тестирование безопасности (Safety Testing)** – тестирование программного продукта с целью определить его способность при использовании оговоренным образом оставаться в рамках приемлемого риска причинения вреда здоровью, бизнесу, программам, собственности или окружающей среде.
- **Тестирование защищенности (Security Testing) –** тестирование с целью оценить защищенность программного продукта от внешних воздействий (от проникновений).На практике зачастую под термином тестирование безопасности понимают в том числе и тестирование защищенности.
- **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).

Формат <кода>

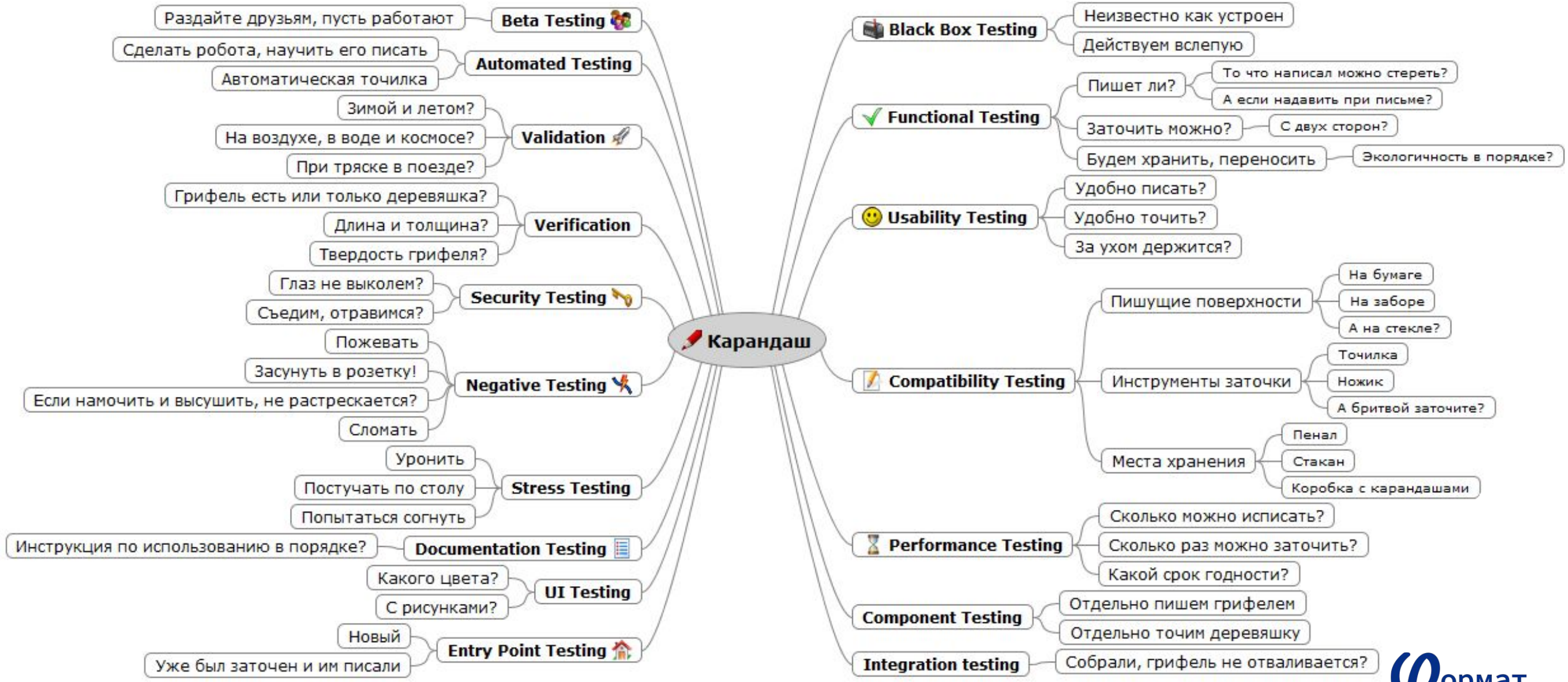
Нефункциональные виды тестирования

- **Тестирования производительности:**
  - **нагрузочное тестирование (Performance and Load testing)** – вид тестирования производительности, проводимый с целью оценки поведения компонента или системы при возрастающей нагрузке, например количестве параллельных пользователей и/или операций, а также определения какую нагрузку может выдержать компонент или система.
  - **стрессовое тестирование (Stress testing)** –вид тестирования производительности, оценивающий систему или компонент на граничных значениях рабочих нагрузок или за их пределами, или же в состоянии ограниченных ресурсов, таких как память или доступ к серверу.
  - **тестирование стабильности и надежности (Stability/ Reliability testing)** –позволяет проверять работоспособность приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки.
  - **объемное тестирование (Volume testing)** – позволяет получить оценку производительности при увеличении объемов данных в базе данных приложения

- Нагрузочное тестирование — при **нормальных** условиях.
- Стрессовое тестирование — при **экстремальных** нагрузках.
- Тестирование стабильности — при **длительной** работе.

Нефункциональные виды тестирования

- **Тестирование Установки (Installation Testing) -** направленно на проверку успешной инсталляции и настройки, а также на обновление или удаление программного обеспечения.
- **Тестирование пользовательского интерфейса (GUITesting**) - тестирование, выполняемое путем взаимодействия с системой через графический интерфейс пользователя
- **Тестирование удобства использования (Usability Testing) -** тестирование с целью определения степени понятности, легкости в изучении и использовании, привлекательности программного продукта для пользователя при условии использования в заданных условиях эксплуатации
- **Конфигурационное тестирование (Configuration Testing) -** специальный вид тестирования, направленный на проверку работы программного обеспечения при различных конфигурациях системы
- **Тестирование на отказ и восстановление (Failover and Recovery Testing) -** Исследование программной системы на предмет восстановления после ошибок, сбоев.
- **Тестирование локализации (localization testing**) — проверка адаптации программного обеспечения для определенной аудитории в соответствии с ее культурными


Карандаш
Beta Testing
Раздайте друзьям, пусть работают
Automated Testing
Сделать робота, научить его писать
Автоматическая точилка
Validation
Зимой и летом?
На воздухе, в воде и космосе?
При тряске в поезде?
Verification
Грифель есть или только деревяшка?
Длина и толщина?
Твердость грифеля?
Security Testing
Глаз не выколем?
Съедим, отравимся?
Negative Testing
Пожевать
Засунуть в розетку!
Если намочить и высушить, не растрескается?
Сломать
Stress Testing
Уронить
Постучать по столу
Попытаться согнуть
Documentation Testing
Инструкция по использованию в порядке?
UI Testing
Какого цвета?
С рисунками?
Entry Point Testing
Новый
Уже был заточен и им писали
Black Box Testing
Неизвестно как устроен
Действуем вслепую
Functional Testing
Пишет ли?
То что написал можно стереть?
А если надавить при письме?
Заточить можно?
С двух сторон?
Будем хранить, переносить
Экологичность в порядке?
Usability Testing
Удобно писать?
Удобно точить?
За ухом держится?
Compatibility Testing
Пишущие поверхности
На бумаге
На заборе
А на стекле?
Инструменты заточки
Точилка
Ножик
А бритвой заточите?
Места хранения
Пенал
Стакан
Коробка с карандашами
Performance Testing
Сколько можно исписать?
Сколько раз можно заточить?
Какой срок годности?
Component Testing
Отдельно пишем грифелем
Отдельно точим деревяшку
Integration testing
Собрали, грифель не отваливается?

Формат <кода>

Тест-анализ и тест-дизайн

Роли в тест дизайне

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

Тест-анализ - процесс поиска и рассмотрения информации, необходимой для тестирования. Обычно это люди со знаниями о системе и процессах, а также документация (требования, спецификации, описания архитектуры и интеграции и т.п).

Проектирование тестов (Тест дизайн, test design) - процесс проектирования и создания тест-кейсов (тестовых случаев/сценариев/дел, case - юр. "дело"), в соответствии с определёнными ранее критериями качества, целями тестирования, критериями приёмки.

Исполняющему роль тест-аналитика необходимо:

- Узнать кто является **причастными сторонами**Выяснить **цель проекта/доработки**: для каких целей создан Продукт/Система, кто и каким образом будет использовать.
- Выяснить **суть реализации** (общей или конкретного инкремента)
- Определить **тестовое покрытие** (что будем тестировать и в каких объёмах) и необходимые виды тестирования.
- (опционально) Определить **Риски тестирования**
- (опционально) Составить **Матрицу трассировки требований**

Формат <кода>

Техники тест-дизайна

- Тестирование на основе классов эквивалентности (equivalence partitioning)
- Техника анализа граничных значений (boundary value testing)
- Тестирование на основе состояний и переходов (State-Transition Testing)
- Тестирование на основе таблицы принятия решений (Decision Table Testing)
- Попарное тестирование (pairwise testing)
- Сценарий использования (Use Case Testing)
- Предугадывание ошибки (Error Guessing - EG)
- Исследовательское тестирование (Exploratory Testing)
- Исчерпывающее тестирование (Exhaustive Testing — ET)

Тестирование на основе классов эквивалентности (equivalence partitioning)

- **Тестирование на основе классов эквивалентности (equivalence partitioning)** — это техника, основанная на методе чёрного ящика, при которой мы разделяем функционал (часто диапазон возможных вводимых значений) на группы эквивалентных по своему влиянию на систему значений.

Тестовые данные разбиваются на определенные классы допустимых значений. В рамках каждого класса выполнение теста с любым значением тестовых данных приводит к эквивалентному результату. После определения классов необходимо выполнить хотя бы один тест в каждом классе.

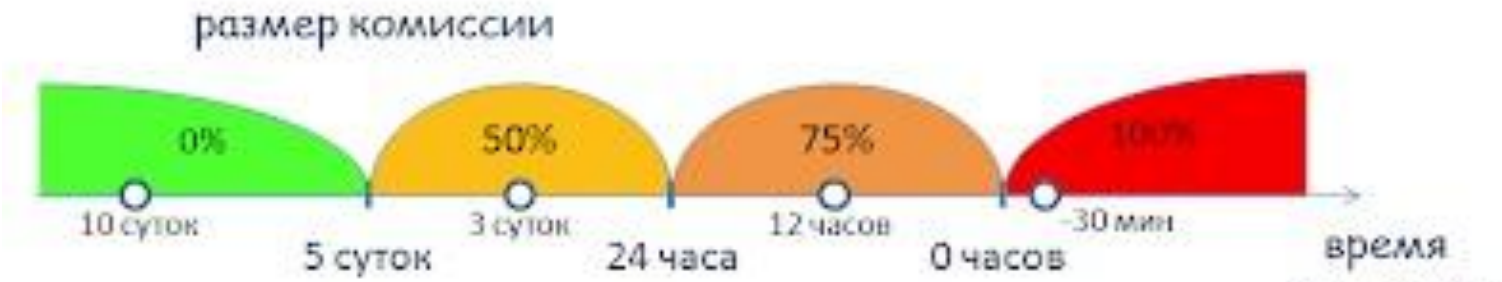
Алгоритм использования техники:

- Необходимо определить класс эквивалентности. Это главный шаг техники. От него во многом зависит эффективность её применения.
- Затем нужно выбрать одного представителя от каждого класса. На этом шаге из каждого эквивалентного набора тестов мы выбираем один тест.
- Нужно выполнить тесты. На этом шаге мы выполняем тесты от каждого класса эквивалентности.

Тестирование на основе классов эквивалентности. Пример

Пример: функция подсчета комиссии при отмене бронирования авиабилетов. Размер комиссии зависит от времени до вылета, когда совершена отмена:

- За 5 суток до вылета комиссия составляет 0%.
- Меньше 5 суток, но больше 24 часов – 50%.
- Меньше 24 часов, но до вылета – 75%.
- После вылета – 100%.



Формат <кода>

Тестирование на основе классов эквивалентности. Пример

Шаги:

1. Определим классы эквивалентности:

(для каждого теста из этих классов мы ожидаем получить одинаковый результат):

- 1 класс: время до вылета > 5 суток.
- 2 класс: 24 часа < время до вылета < 5 суток.
- 3 класс: 0 часов < время до вылета < 24 часа.
- 4 класс: время до вылета < 0 часов (вылет уже состоялся).

2. Выберем представителя от каждого класса:

- время до вылета = 10 суток (тест из 1-го класса).
- время до вылета = 3 суток (тест из 2-го класса).
- время до вылета = 12 часов (тест из 3-го класса).
- время до вылета = -30 мин (тест из 4-го класса).

3. Выполним тесты:

- Отменим бронь за 10 суток до вылета и проверим, что комиссия составила 0%.
- Отменим бронь за 3 суток до вылета и проверим, что комиссия составила 50%.
- Отменим бронь за 12 часов до вылета и проверим, что комиссия составила 75%.

Техника анализа граничных значений (boundary value testing)

Техника анализа граничных значений (boundary value testing) — это техника проверки поведения продукта на крайних (граничных) значениях входных данных.

Техника граничных значений основана на предположении, что большинство ошибок может возникнуть на границах эквивалентных классов. Она тесно связана с вышеописанной техникой эквивалентного разбиения, из-за чего часто используется с ней в паре.

Алгоритм использования техники анализа граничных значений:

- Во-первых, нужно выделить классы эквивалентности. Опять же, это очень важный шаг и от правильности разбиения на классы эквивалентности зависит эффективность тестов граничных значений.
- Далее нужно определить граничные значения этих классов.
- Нам нужно понять, к какому классу будет относиться каждая граница.
- Для каждой границы нам нужно провести тесты по проверке значения до границы, на границе, и сразу после границы.

Техника анализа граничных значений. Пример

Пример: функция подсчета комиссии при отмене бронирования авиабилетов. Размер комиссии зависит от времени до вылета, когда совершена отмена:

- За 5 суток до вылета комиссия составляет 0%.
- Меньше 5 суток, но больше 24 часов – 50%.
- Меньше 24 часов, но до вылета – 75%.
- После вылета – 100%.

Формат <кода>

Техника анализа граничных значений. Пример

Шаги:

1. Выделим классы эквивалентности:

- 1 класс: время до вылета > 5 суток.
- 2 класс: 24 часа < время до вылета < 5 суток.
- 3 класс: 0 часов < время до вылета < 24 час.
- 4 класс: время до вылета < 0 часов (вылет уже состоялся).

2. Определим границы:

- 5 суток.
- 24 часа.
- 0 часов.

Техника анализа граничных значений. Пример

3. Определим, к какому классу относятся границы:

- Примечание: На этом шаге был спорный момент, на который нужно обратить внимание. При составлении задачи я не подумал, какая логика должна быть на самих границах. Это типичный пример того, как составители требований не задумываются о границах. И поэтому программист может трактовать их по-своему.
- 5 суток – ко 2-му классу.
- 24 часа – к о2-му классу.
- 0 часов – к 4-му классу.

4. Протестируем значения на границах, до и после них:

- Отменим бронь за 5 суток + 1 секунду до вылета (или просто постараемся выполнить бронь как можно ближе к границе, но слева от нее) и проверим, что комиссия равна 0%.
- Отменим бронь ровно за 5 суток до вылета и проверим, что комиссия равна 50%.
- Отменим бронь за 5 суток – 1 секунду до вылета и проверим, что комиссия равна 50%.
- Отменим бронь за 24 часа + 1 секунду до вылета и проверим, что комиссия равна 50%.
- Отменим бронь ровно за 24 часа до вылета и проверим, что комиссия равна 50%.
- Отменим бронь за 24 часа — 1 секунду до вылета и проверим, что комиссия равна 75%.
- Отменим бронь за 1 секунду до вылета и проверим, что комиссия равна 75%.
- Отменим бронь ровно во время вылета и проверим, что комиссия равна 100%.

Формат <кода>

Тестирование на основе состояний и переходов (State-Transition Testing)

- **Тестирование на основе состояний и переходов (State-Transition Testing)** — Техника для визуализации ТЗ. Она наглядно показывает, как некий объект переходит из одного состояния в другое.

Как рисовать диаграмму

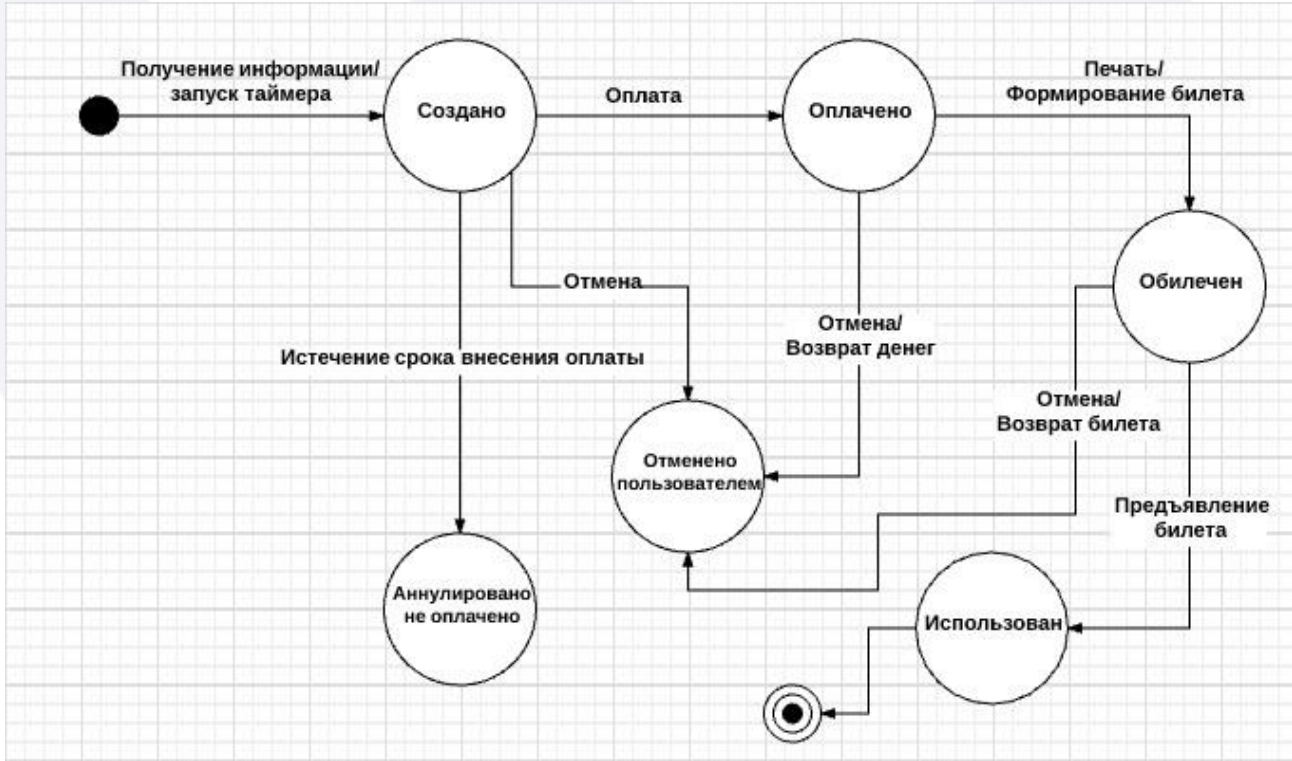
1. Вы выбираете объект в своём проекте

2. Думаете, какие у него состояния. Они не должны пересекаться, то есть: объект не может быть разом в двух состояниях, и при этом он всегда хоть в каком-то одном есть

3. Рисуете эти состояния кружочками.

4. Соединяете их стрелочками. Стрелочки - это действия, их надо подписать.

5. Смотрите, что получилось и анализируете, есть ли у объекта другие состояния? А другие действия между текущими состояниями? Переход на шаг 2.



Тестирование на основе состояний и переходов

Может быть выбран один из 4 вариантов создания тест-кейсов:

- Создать наборы тест-кейсов так, чтобы **все состояния** были пройдены хотя бы по одному разу. В одном тест-кейсе может быть описан переход через несколько состояний. Это довольно слабый уровень тестового покрытия.
- Создать наборы тест-кейсов так, чтобы **все события** были инициированы хотя бы по одному разу. Тест-кейсы, которые покрывают все события в то же время покрывают и все состояния. Снова слабый уровень тестового покрытия.
- Создать наборы тест-кейсов так, чтобы **все пути** были пройдены хотя бы по одному разу. Такой способ хорош с точки зрения тестового покрытия, однако практически не осуществим. Если диаграмма имеет циклы, то количество возможных путей может оказаться бесконечным.
- Создать наборы тест-кейсов так, чтобы **все переходы** были выполнены хотя бы по одному разу. Этот способ обеспечивает хороший уровень тестового покрытия, поэтому рекомендуется использовать именно его.

Тестирование на основе состояний и переходов

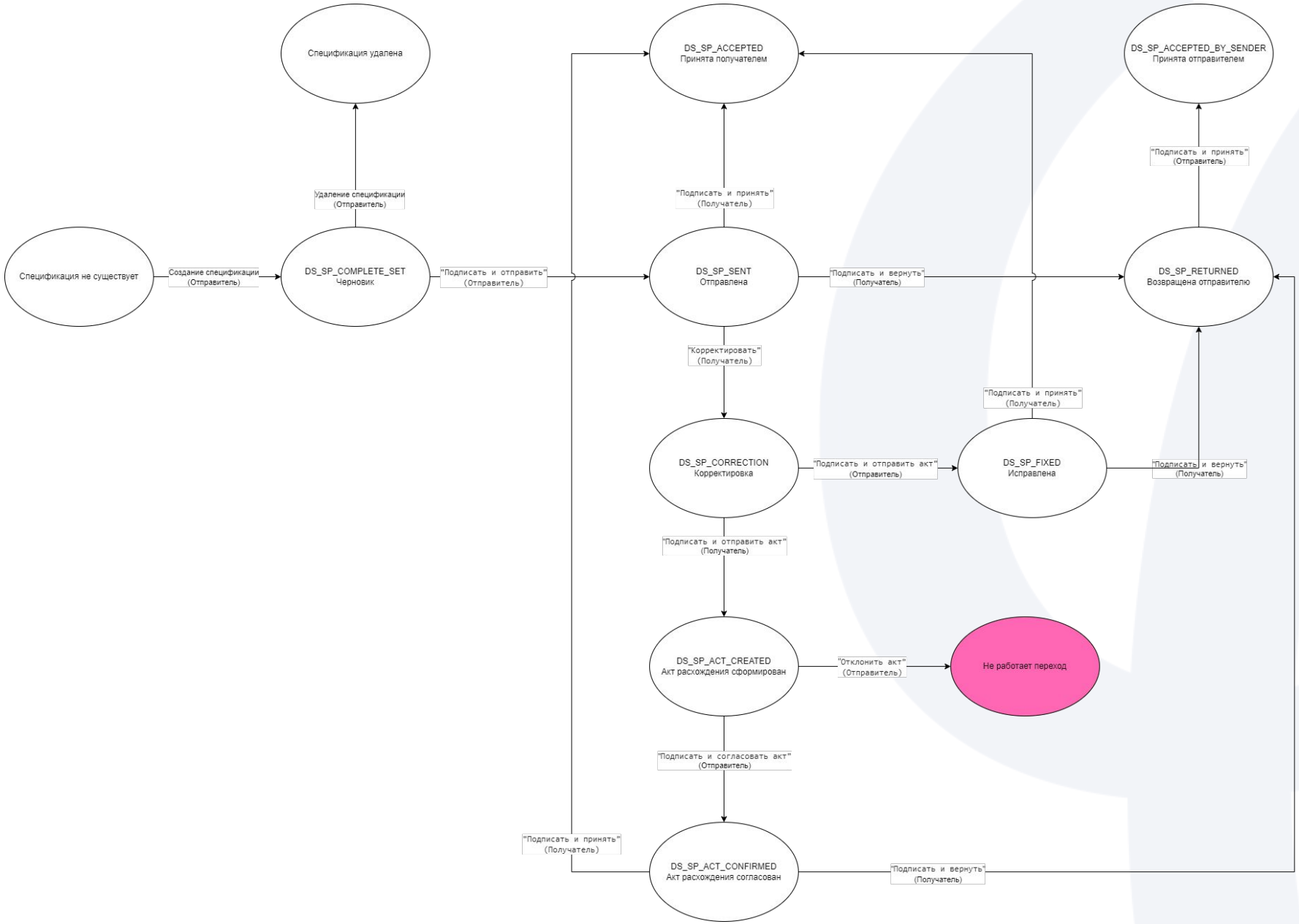
Плюсы подхода

- Красиво выглядит
- Позволяет увидеть, что мы упустили
- Наглядно видим весь маршрут нашего объекта
- Можем понять, что упускаем

Минусы подхода

- Слишком насыщенная карта — разбиваем на несколько маленьких.
- Сложно поддерживать — нужна ли она вообще?

Тестирование на основе состояний и переходов. Пример схемы



Тестирование на основе состояний и переходов. Пример таблицы

Из / В	Спецификация не существует	DS_SP_COMPLETE_SET	DS_SP_SENT	DS_SP_ACCEPTED	DS_SP_RETURNED	DS_SP_ACCEPTED_BY_SENDER	DS_SP_CORRECTION	DS_SP_FIXED	DS_SP_ACT_CREATED	DS_SP_ACT_CONFIRMED
Спецификация не существует	-	Создание спецификации (Отправитель)	-	-	-	-	-	-	-	-
DS_SP_COMPLETE_SET	Удаление спецификации (Отправитель)	-	"Подписать и отправить" (Отправитель)	-	-	-	-	-	-	-
DS_SP_SENT	-	-	-	"Подписать и принять" (Получатель)	"Подписать и вернуть" (Получатель)	-	"Корректировать" (Получатель)	-	-	-
DS_SP_ACCEPTED	-	-	-	-	-	-	-	-	-	-
DS_SP_RETURNED	-	-	-	-	-	"Подписать и принять" (Отправитель)	-	-	-	-
DS_SP_ACCEPTED_BY_SENDER	-	-	-	-	-	-	-	-	-	-
DS_SP_CORRECTION	-	-	-	-	-	-	-	"Подписать и отправить акт" (Отправитель)	"Подписать и отправить акт" (Получатель)	-
DS_SP_FIXED	-	-	-	"Подписать и принять" (Получатель)	"Подписать и вернуть" (Получатель)	-	-	-	-	-
DS_SP_ACT_CREATED	-	-	-	-	-	-	-	-	-	"Подписать и согласовать акт" (Отправитель)
DS_SP_ACT_CONFIRMED	-	-	-	"Подписать и принять" (Получатель)	"Подписать и вернуть" (Получатель)	-	-	-	-	-

Таблицы принятия решений (Decision Table Testing)

Таблицы принятия решений (Decision Table Testing) — техника тестирования, основанная на методе чёрного ящика, которая применяется для систем со сложной логикой. Способ компактного представления модели со сложной логикой. Техника, помогающая наглядно изобразить комбинаторику условий из ТЗ

	Значения	Правило 1	Правило 2	…	Правило N
Условия					
Условие 1	Значение1, Значение 2				
Условие 1	Значение1, Значение 2				
…					
Условие N					
Действия					
Действие 1					
Действие 2					
…					
Действие N					

Как составлять таблицу:

- По горизонтали — выписываем условия, которые влияют на результат. А чуть ниже — сам результат, в оригинале Action — действие, которое нужно выполнить.
- По вертикали — правила: конкретная комбинация входных условий.

Техника выполнения:

1. Определить все условия
2. Составить все возможные комбинации условий
3. Убрать лишние комбинации. Удаляются те, в которых изменение значений никак не влияет на получаемый результат (Don’t care — DC)
4. Определить действия
5. Создать тест-кейсы для каждой комбинации

Формат <кода>

Таблицы принятия решений

Плюсы подхода

- **Наглядность**
- **Нарисовал таблицу = записал тест-кейсы**
- **Наглядность поможет найти баги в документации**
- **Таблица помогает взглянуть на ТЗ свежим взглядом**, по-новому

Минусы подхода

Особых минусов нет, но таблица не нужна, если:

- Слишком простое условие
- Слишком много входных данных

Таблицы принятия решений. Пример

Условия	Значения	Правила				
Логин	Верный, Неверный	Верный	Верный	Верный	Неверный	Неверный
Пароль	Верный, Неверный	Верный	Верный	Неверный	Неверный	Верный
Код	Верный, Неверный, Не пришел	Верный	Неверный	Не пришел	Не пришел	Не пришел
Действия	Пользователь вошел на сайт Пользователь не авторизован	Пользователь вошел на сайт	Пользователь не авторизован	Пользователь не авторизован	Пользователь не авторизован	Пользователь не авторизован

Условия	Значения	Правила			
Логин	Верный, Неверный	Верный	Неверный	Верный, Неверный	Верный
Пароль	Верный, Неверный	Верный	Верный, Неверный	Неверный	Верный
Код	Верный, Неверный, Не пришел	Верный	Верный, Неверный, Не пришел	Верный, Неверный, Не пришел	Неверный
Действия	Пользователь вошел на сайт Пользователь не авторизован	Пользователь вошел на сайт	Пользователь не авторизован	Пользователь не авторизован	Пользователь не авторизован

Формат <кода>

Попарное тестирование (pairwise testing)

Попарное тестирование (pairwise testing) — это техника формирования наборов тестовых данных из полного набора входных данных в системе, которая позволяет существенно сократить количество тест-кейсов. В которой тестовые сценарии разрабатываются таким образом, чтобы выполнить все возможные отдельные комбинации каждой пары входных параметров

Метод парного тестирования основан на следующей идее: подавляющее большинство багов выявляется тестом, проверяющим либо один параметр, либо сочетание двух. Ошибки, причиной которых явились комбинации трех и более параметров, как правило, значительно менее критичны.

Техника выполнения:

1. Определить параметры (variables)
2. Определить количество значений для каждого параметра (choices for variable)
3. Построить массив, содержащий колонки для каждого параметра и значения в колонках, которые содержать все сочетания значений этих параметров друг с другом.
4. Сопоставить полученный ортогональный массив с целью тестирования.
5. Построить тест-кейсы.

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

Формат <кода>

Тестирование на основе сценарий использования (Use Case Testing)

Тестирование на основе сценарий использования (Use Case Testing) — Use Case описывает сценарий взаимодействия двух и более участников (как правило — пользователя и системы). Пользователем может выступать как человек, так и другая система. Варианты использования описываются с точки зрения пользователя, а не системы. Внутренние работы по поддержанию работоспособности системы не являются частью варианта использования.

Рекомендации по созданию тест-кейсов на основе вариантов использования:

1. Начать с валидных данных и наиболее частых сценариев.
2. Проверить граничные значения и невалидные значения (с использованием ранее рассмотренных техник).
3. Редко используемые сценарии, крайне важные для системы (так называемая “Остановка ядерного реактора” Shut Down The Nuclear Reactor)
4. Тесты на каждое ветку-альтернативу (Extension) каждого шага
5. Попробовать выполнить операцию в непривычном порядке
6. Извратить предусловие, если это действительно может произойти
7. Если транзакция имеет циклы, запустите ее в цикле, и не один-два раза — будьте жестче
8. Найти очень долгий и извилистый путь и пройдите по нему
9. Если ожидается, что транзакция будет выполняться в логичном порядке, попробовать выполнить ее в обратном порядке (например заполнить поля не сверху вниз, а снизу вверх)

Предугадывание ошибки (Error Guessing - EG)

Предугадывание ошибки (Error Guessing - EG) - это способ предотвращения ошибок, дефектов и отказов, основанный на знаниях тестировщика, включающих:

- — Историю работы приложения в прошлом.
- — Наиболее вероятные типы дефектов, допускаемых при разработке.
- — Типы дефектов, которые были обнаружены в схожих приложениях.

Преимущества:

1. Эта проверка эффективна в качестве дополнения к другим техникам.
2. Выявляет тестовые случаи, которые “никогда не должны случиться”.

Недостатки:

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

Формат <кода>

Исследовательское тестирование (Exploratory Testing)

Исследовательское тестирование (Exploratory Testing) - Это достаточно гибкое тестирование, которое говорит нам о том, что тест-кейсы и чек-листы создаются, выполняются, анализируются и оцениваются динамически во время выполнения тестов.

Виды:

- чит-листы
- сессионное тестирование
- парное тестирование
- тест-туры Джеймса Виттакера (James A. Whittaker)

Сессионное тестирование (Session-based testing)

- **Сессионное тестирование (Session-based testing) -** это метод тестирования программного обеспечения, целью которого является сочетание подотчетности и исследовательского тестирования для обеспечения быстрого обнаружения дефектов, творческого проектирования тестов на лету, управленческого контроля и отчетности по метрикам

Элементы session-based testing

1. Миссия
2. Устав
3. Сессия
4. Отчет о сеансе
5. Разбор полетов
6. Анализ результатов

Парное тестирование (Pair testing)

- **Парное тестирование (Pair testing)** – Это когда несколько человек (два тестировщика, разработчик и тестировщик, или конечный пользователь и тестировщик), работают вместе над поиском дефектов.

Тест-туры

Туры – это идеи и инструкции по исследованию программного продукта, объединенные определённой общей темой или целью.

Туры:

1. Бизнес-районы
2. Исторические районы
3. Туристические районы
4. Развлекательные районы
5. Районы отелей
6. Захудалые районы

Тест-туры

Бизнес-районы:

- Тур по путеводителю
- Денежный тур
- Тур по отметкам
- Интеллектуальный тур
- Тур службы доставки
- Тур «после работы»
- Тур уборщика

Исторические районы:

- Тур «плохие соседи»
- Музейный тур
- Тур прежней версии

Туристические районы:

- Тур коллекционера
- Тур одинокого бизнесмена
- Тур супермодели
- Тур «тестируй одно, другое – бесплатно» (Тур шопоголика)
- Тур шотландского паба

Тест-туры

Развлекательные районы:

- Тур актера второго плана
- Тур по задней аллее (Тур по темным переулкам)
- Тур любителя ночной жизни

Районы отелей:

- Тур, отмененный из-за дождя
- Тур лежебоки

Захудалые районы:

- Тур диверсанта
- Антиобщественный тур
- Тур навязчивости и даже одержимости

Формат <кода>

Исчерпывающее тестирование (Exhaustive Testing — ET)

Исчерпывающее тестирование (Exhaustive Testing — ET) — это крайний случай. В пределах этой техники Вы должны проверить все возможные комбинации входных значений, и в принципе, это должно найти все проблемы. На практике применение этого метода не представляется возможным из-за огромного количества входных значений

Знания и личные качества тестировщика

1. Усидчивость и терпение
2. Внимательность
3. Любопытство и пытливый ум
4. Гибкость и способность быстро переключаться
5. Способность расставлять приоритеты и отсутствие перфекционизма
6. Умение четко донести свою логику до других через текст
7. Тестировщик должен быть командным игроком
8. Тестировщик должен думать как пользователь
9. Ну и конечно знание основ тестирования

Кодекс этики

- **ОБЩЕСТВО** – тестировщики П.О. должны действовать согласно интересам общества
- **КЛИЕНТ И РАБОТОДАТЕЛЬ** – тестировщики П.О. должны действовать согласно интересам клиента и работодателя, если они не противоречат интересам общества.
- **ПРОДУКТ** – тестировщики П.О. должны быть уверены в том, что компоненты, которые они проверяют (в тестируемых продуктах или системах), соответствуют наивысшим возможным профессиональным стандартам.
- **ОЦЕНКИ** – тестировщики П.О. должны поддерживать целостность и независимость своих профессиональных оценок.
- **УПРАВЛЕНИЕ** – руководители тестирования П.О. и ведущие специалисты должны присоединяться и продвигать этические подходы к управлению тестированием программного обеспечения.
- **ПРОФЕССИЯ** – тестировщики П.О. должны поднимать престиж и репутацию своей профессии в интересах общества.
- **КОЛЛЕГИ** – тестировщики П.О. должны быть справедливыми, оказывать поддержку своим коллегам, и содействовать сотрудничеству с разработчиками программного обеспечения.
- **ЛИЧНАЯ ОТВЕТСТВЕННОСТЬ** – тестировщики П.О. должны постоянно учиться

Формат <кода>

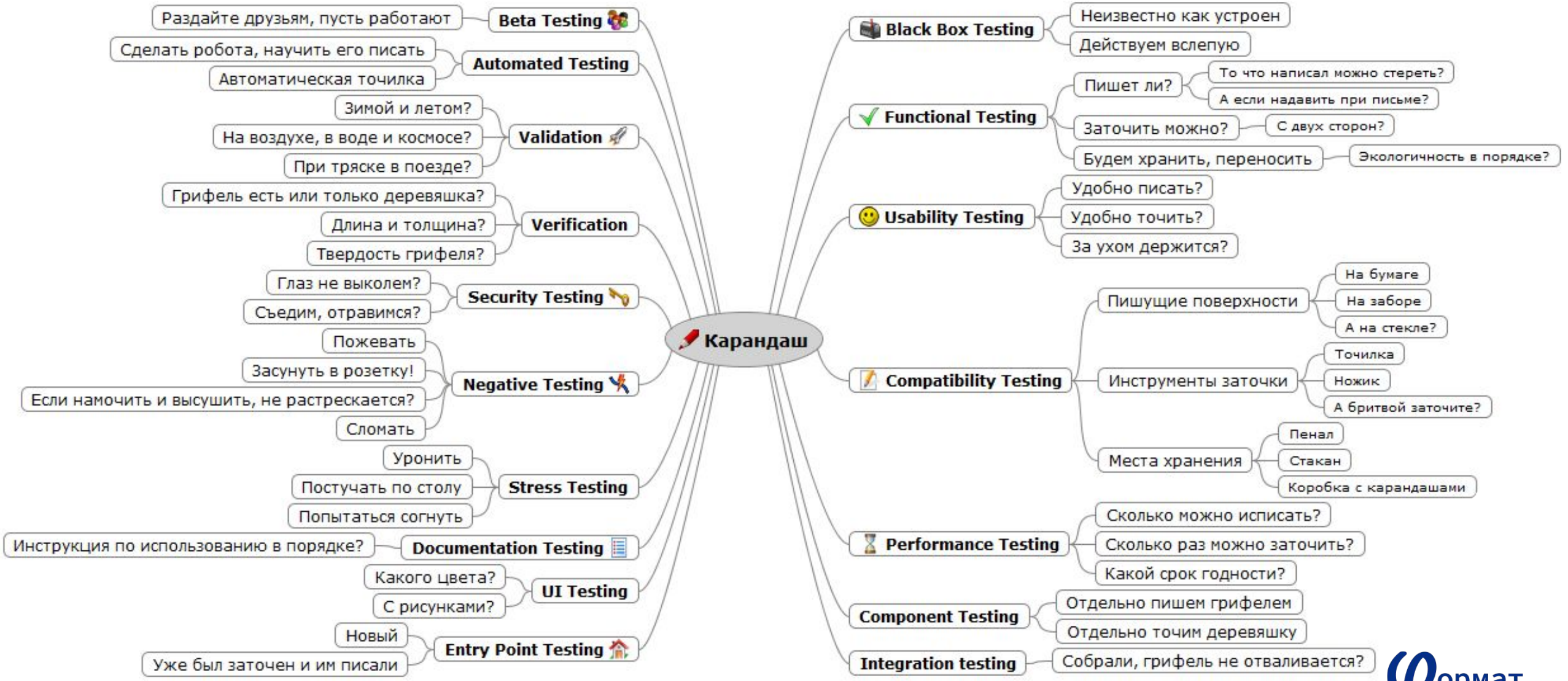
Практическая часть

Практика. Типы (виды) тестирования

1. Классификация по запуску кода на исполнение
2. Классификация по доступу к коду и архитектуре (Знанию системы)
3. Классификация по уровню детализации приложения:
4. Классификация по степени автоматизации
5. Классификация по принципам работы с приложением
6. Виды тестирования в зависимости от подготовленности:
7. Связанные с изменениями виды тестирования
8. Классификация в зависимости от времени проведения
9. Классификация в зависимости от целей тестирования

Формат <кода>

Практика. Типы (виды) тестирования



Формат <кода>

1. Классификация по запуску кода на исполнение:
   - Статическое тестирование
   - Динамическое тестирование
2. Классификация по доступу к коду и архитектуре (Знанию системы):
   - Тестирование белого ящика (white box)
   - Тестирование чёрного ящика (black box)
   - Тестирование серого ящика (grey box)
3. Классификация по уровню детализации приложения:
   - Компонентное (модульное) теестирование (component/unit testing)
   - Интеграционное тестирование (integration testing)
   - Системное тестирование (system/end-to-end testing)
   - Приёмочное тестирование
4. Классификация по степени автоматизации:
   - Ручное тестирование (manual testing)
   - Автоматизированное тестирование (automated testing)
   - Полуавтоматизированное тестирование (semiautomated testing)
5. Классификация по принципам работы с приложением
   - Позитивное тестирование
   - Негативное тестирование
6. Виды тестирования в зависимости от подготовленности:
   - Интуитивное тестирование
   - Исследовательское тестирование
   - Тестирование по документации

Формат <кода>

7. Связанные с изменениями виды тестирования:
   - Подтверждающее тестирование (Re-testing)
   - Дымовое тестирование (smoke test)
   - Санитарное тестирование (Sanity Testing)
   - Регрессионное тестирование (regression testing)
   - Тестирование сборки (Build Verification Test)
8. Классификация в зависимости от времени проведения:
   - Альфа-тестирование
   - Бета-тестирование
   - Гамма-тестирование(gammatesting)
9. Классификация в зависимости от целей тестирования:
   - Функциональные виды:
     1. Функциональное тестирование (functional testing)
     2. Тестирование безопасности (Safety Testing)
     3. Тестирование защищенности (Security Testing)
     4. Тестирование взаимодействия (Interoperability Testing)
   - Нефункциональное тестирование (non-functional testing):
     1. Тестирование производительности (performance testing)
     2. тестирование Установки (installation testing)
     3. Тестирование интерфейса (GUI/UI testing)
     4. Тестирование удобства использования (usability testing)
     5. Конфигурационное тестирование (Configuration Testing)
     6. Тестирование на отказ и восстановление (Failover and Recovery Testing)
     7. Тестирование локализации (localization testing)

Практика. Написание Чек-листа

Чек-лист (check list) — это документ, который описывает что должно быть протестировано. Чек-лист может быть абсолютно разного уровня детализации.

Чек-лист должен обладать рядом важных свойств:

- Логичность
- Последовательность и структурированность
- Полнота и неизбыточность

Правила составления чек-листов:

- Одна операция
- Пункты чек-листа - это минимальные полные операции
- Пункты пишутся в утвердительной форме

Практика. Написание Чек-листа

Чек-лист “Обратная связь”

1. Отправка сообщения с пустым полем “Имя”
2. Отправка сообщения с именем на кириллице, введенным в поле “Имя”
3. Отправка сообщения с именем на латинице, введенным в поле “Имя”
4. Отправка сообщения с именем из цифр, введенным в поле “Имя”
5. Отправка сообщения с именем из одного символа, введенным в поле “Имя”
6. Отправка сообщения с пустым полем “Email”
7. Отправка сообщения с корректным email, введенным в поле “Email”
8. Отправка сообщения с пустым полем “Сообщение”
9. Отправка сообщения с текстом на кириллице, введенным в поле “Сообщение”
10. Отправка сообщения с текстом на латинице, введенным в поле “Сообщение”

Одинаковый соседний перцентиль

Поиск и исправление одинаковых соседних перцентилей в AGP (и ATP)
Можем закрыть код.
Можем закрыть перцентили.
Можем вызвать другие ошибки
файлы: https://drive.google.com/open?id=1V4G7p_mQnB083WCkqdD3OeZdptKQihpn

Аббр.:	Регулярность прохождения:	Тип чек-листа	Статус:	Готовность %:	Трудозатратность:
ОдСП	Не указано	Проверки и результаты	В процессе создания	0	0.00

Ошибка в AGP

6

Находим строки где AGP равняется значению одинаковых перцентилей
Группируем строки по компаниям.
Удаляем строки из компании, где их больше, кроме последней.
Переходим к следующей компании.
Потом удаляем последнии строки в компаниях.

НОМЕР	ПРОВЕРКА	ОЖИДАЕМЫЙ РЕЗУЛЬТАТ
0 ОдСП 1	Загружаем файл где исправляется ошибка в одинаковом перцентиле AGP	ошибка исправилась. строки вызывающие ошибку скрыты (скрыты строки где AGP = ошибочному перцентилю)
0 ОдСП 2	Загружаем файл где исправляется ошибка в одинаковом перцентиле AGP, вызывает скрытие 90 перцентиля	ошибка исправилась. строки вызывающие ошибку скрыты (скрыты строки где AGP = ошибочному перцентилю). 90 перцентиль закрыт
0 ОдСП 3	Загружаем файл где исправление ошибки в одинаковом перцентиле AGP вызывает закрытие компании	ошибка исправилась. строки вызывающие ошибку скрыты (скрыты строки где AGP = ошибочному перцентилю).
0 ОдСП 4	Загружаем файл где исправление ошибки в одинаковом перцентиле AGP вызывает закрытие кода	не подобраны данные (заказчик считает что данный кейс не возможен)
0 ОдСП 5	Загружаем файл где исправление ошибки в одинаковом перцентиле AGP, вызывает другой тип ошибок	не подобраны данные
0 ОдСП 6	Загружаем файл, где не исправляется ошибка в одинаковом перцентиле AGP	не подобраны данные (заказчик считает что данный кейс не возможен)

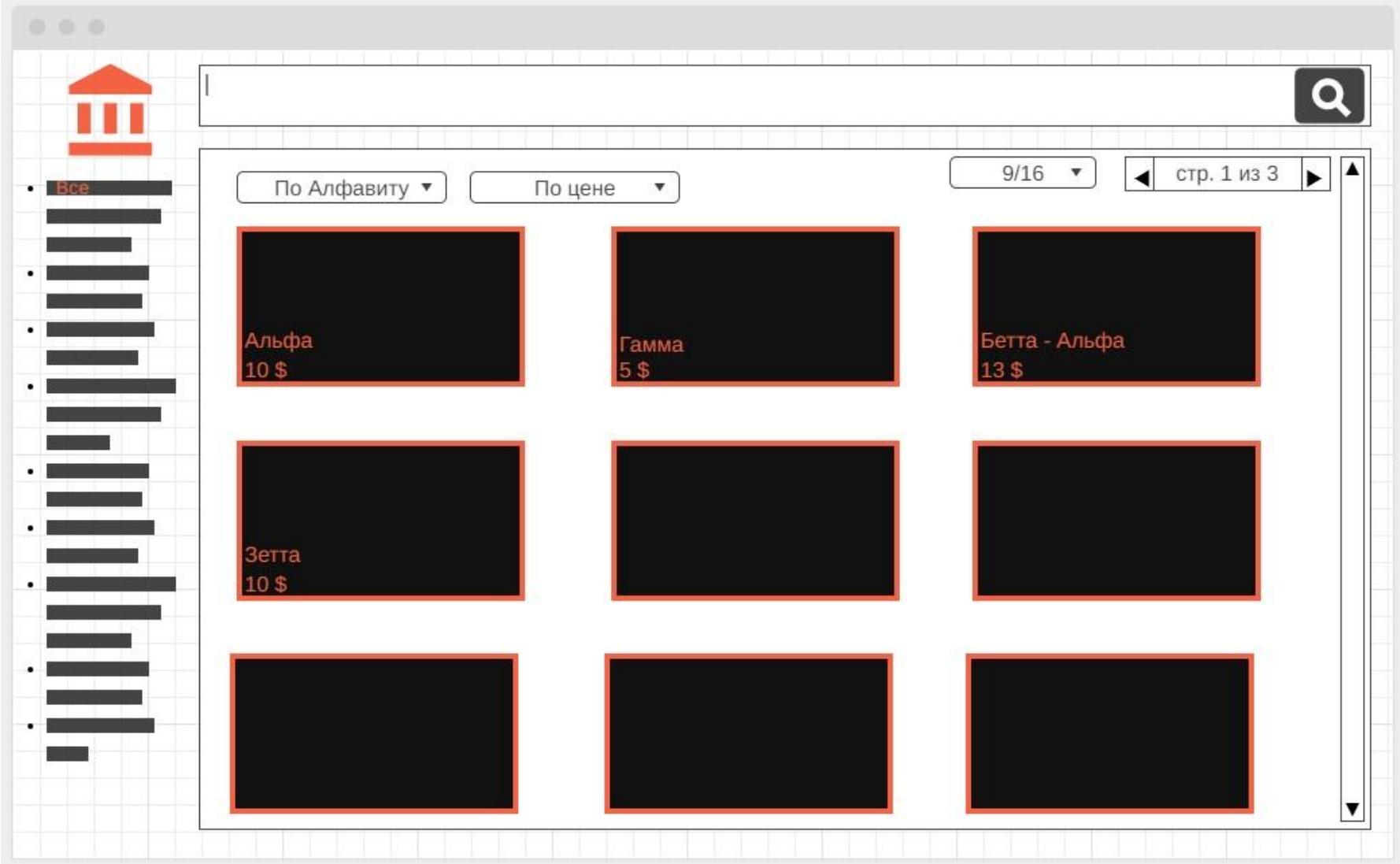
1) Список Каталогов:
- Каталоги отображаются по алфавиту
- Каталог "Все" отображается первым
- При выборе каталога - отображаются товары входящие в него

2) Список товаров:
- Отображается 9 товаров по умолчанию
- Сортировка по алфавиту, по возрастанию по умолчанию

3) Сортировка:
- По алфавиту. По возрастани, По убыванию
- По цене. По возрастани, По убыванию
- Можно выбрать только один вариант
- При выборе сортировки отображается знак сортировки (треугольник)

4) Пейджинация:
- Можно выбрать отображение 9 или 16 товаров на страницу
- Можно переключать страницы
- Отображается номер страницы и общее число страниц

5) Поиск:
- Можно искать по слову целиком
- Можно искать по части слова
- Поиск происходит по нажатию иконки поиска. И по нажатию клавиши "Enter" на клавиатуре



Формат кода

Практика. Написание Тест-Кейса

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

Атрибуты тест-кейса:

- **Уникальный идентификатор тест-кейса**
- **Название**
- **Предусловия**
- **Шаги**
- **Ожидаемый результат**
- **Постусловия**
- **Приоритет**
- **Приложения**

Формат <кода>

- **ID**: FWSF-1. (Лучше использовать числа в возрастающем порядке. FWSF = FootWear Search Functionality. Попробуйте придумать комбинацию букв, имеющую отношение к проекту или функции, которую вы собираетесь тестировать).
- **Заголовок**: Проверить результаты поиска с корректными входными данными. (Узнать, какие значения допустимы, мы можем в требованиях).
- **Предусловия**: Нужно иметь предварительно настроенные продукты из разных категорий, отображаемые на сайте. (Для проверки функциональности нам необходимо иметь элементы, доступные для поиска. Вы можете настроить это в панели администратора или в базе данных).
- **Шаги**:
  1. Откройте домашнюю страницу. (Ссылка не обязательна, ее наличие может затруднить поддержку тест-кейса в будущем).
  2. Введите в поле поиска ключевое слово, связанное с названием доступного продукта.
  3. Выполните поиск, кликнув значок поиска или нажав Enter.
  4. Проверьте результаты.
- **Ожидаемый результат**: На странице результатов поиска отображаются все

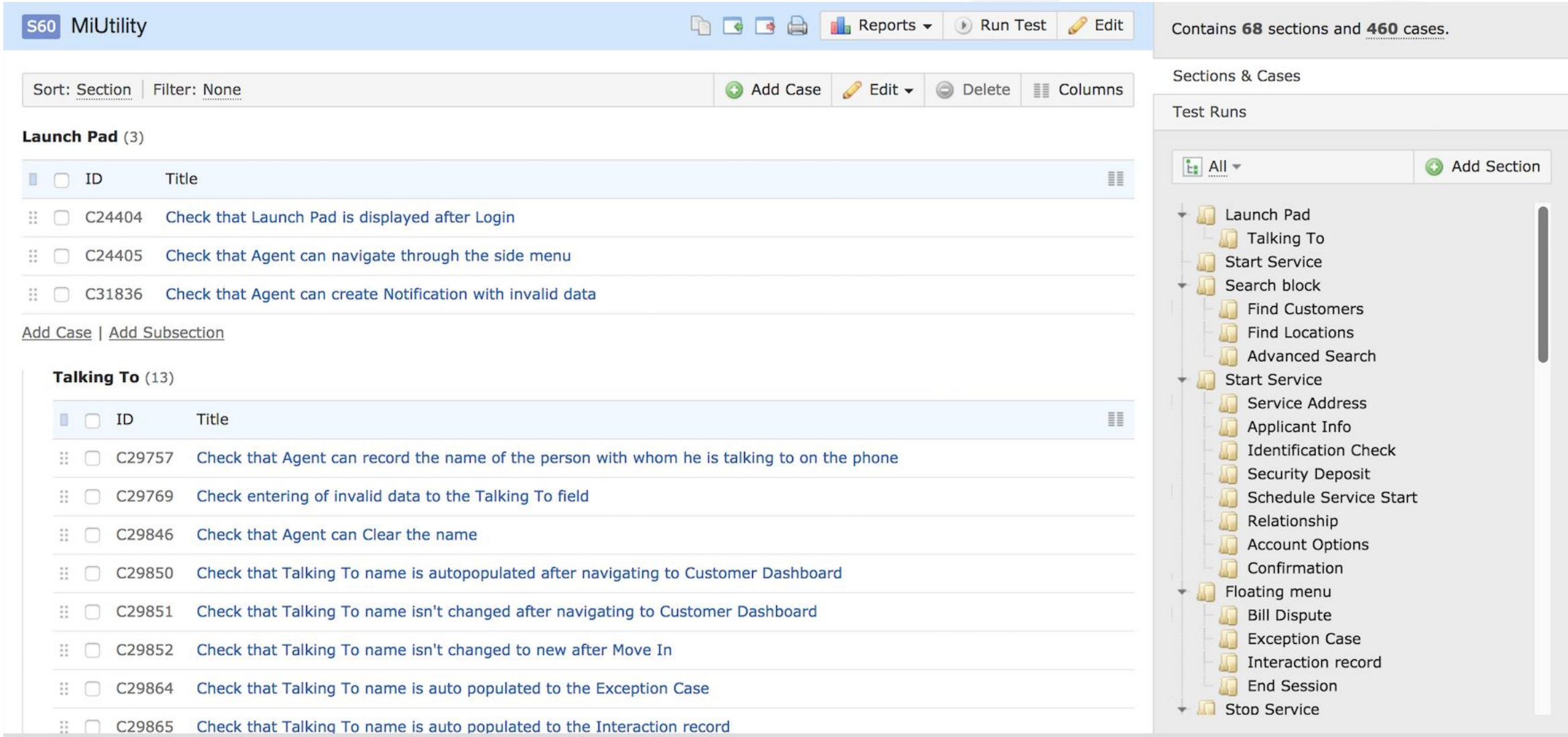
Формат <кода>

Номер	1
Заголовок	Отправка сообщения через форму обратной связи на странице "Контакты"
Предусловие	Открыта главная страница сайта victorz.ru. Есть доступ к почте администратора сайта victorz.ru

Шаг	**Ожидаемый результат**
В верхнем меню сайта нажать на ссылку "Контакты"	Открылась страница "Контакты"
Ввести значение в поле "Ваше имя" состоящее из латинских букв, кириллицы	В поле "Ваше имя" отображается введённое имя
Ввести корректный email в поле "Ваш e-mail"	В поле "Ваш e-mail" отображается введённый email
Ввести в поле "Тема" значение состоящее из латинских букв, кириллицы, спецсимволов и чисел	В поле "Тема" отображается введённый текст
Ввести в поле "Сообщение" значение состоящее из латинских букв, кириллицы, спецсимволов и чисел	В поле "Сообщение" отображается введённый текст
Ввести в поле капчи требуемое капчей значение	В поле капчи отображается введённое значение
Нажать под заполняемой формой на кнопку "Отправить"	Под кнопкой «Отправить» появился текст "Спасибо. Ваше сообщение было отправлено." Все заполненные поля очищены.
Проверить почту администратора сайта	На почту пришло сообщение, отправленное с сайта через форму обратной связи и содержащее в теле сообщения данные введённые на шагах 1-5.

Номер	1
Заголовок	Отправить сообщение через форму обратной связи (Указываем, что проверяем или что делаем?)
Предусловие	Перейти на главную страницу сайта victorz.ru (Это не предусловие, а описание шага)

Шаг	**Ожидаемый результат**
Нажать на ссылку "Контакты" (Где она находится?)	Открылась страница (Какая?)
Ввести имя в поле "Ваше имя" (Какие символы вводить?)	(Ничего не указано в ожидаемом результате, что должно произойти?)
Ввести email в поле "Ваш e-mail" (корректный или некорректный?)	В поле отображается email (Какой? Введённый? В каком поле отображается?)
Ввести в поле значение, состоящее из латинских букв, кириллицы, спецсимволов и чисел (В какое поле?)	В поле "Тема" отображается текст (Какой?)
Ввести в поле "Сообщение" текст (Какие символы вводить?)	Видим в поле "Сообщение" введённый текст (Видим или отображается?)
Вводим в поле капчи требуемое капчей значение (Помните только безличные глаголы — Ввести).	В поле капчи будет введённое значение (Что будет делать? Танцевать?)
Нажать под заполняемой формой на кнопку (На какую?)	Появился текст "Спасибо. Ваше сообщение было отправлено." (Где появится?)
(Последний шаг не заполнен, а это неправильно, так как мы не проверим действительно ли работает отправка писем через форму обратной связи)	


S60 MiUtility
Reports
Run Test
Edit
Contains 68 sections and 460 cases.
Sort: Section | Filter: None
Add Case
Edit
Delete
Columns
Launch Pad (3)
ID
Title
C24404 Check that Launch Pad is displayed after Login
C24405 Check that Agent can navigate through the side menu
C31836 Check that Agent can create Notification with invalid data
Add Case | Add Subsection
Talking To (13)
ID
Title
C29757 Check that Agent can record the name of the person with whom he is talking to on the phone
C29769 Check entering of invalid data to the Talking To field
C29846 Check that Agent can Clear the name
C29850 Check that Talking To name is autopopulated after navigating to Customer Dashboard
C29851 Check that Talking To name isn't changed after navigating to Customer Dashboard
C29852 Check that Talking To name isn't changed to new after Move In
C29864 Check that Talking To name is auto populated to the Exception Case
C29865 Check that Talking To name is auto populated to the Interaction record
Sections & Cases
Test Runs
All
Add Section
Launch Pad
Talking To
Start Service
Search block
Find Customers
Find Locations
Advanced Search
Start Service
Service Address
Applicant Info
Identification Check
Security Deposit
Schedule Service Start
Relationship
Account Options
Confirmation
Floating menu
Bill Dispute
Exception Case
Interaction record
End Session
Stop Service

Формат <кода>

C19224 Check creation of relationship with invalid SSN/FEIN

MiUtility › Start Service › Relationship

Successfully updated the test case.

Type	Priority	Estimate	References
Other	Medium	None	AI-49

Preconditions

- User has logged into the MiAgent
- User has started service
- User has gone to Relationship step

Steps

	Step	Expected Result
1	Click on "Yes" button near "Would you like to add a Relationship to the account?" text	The Relationship form is displayed
2	Enter valid Name and select Relationship type	The following relationship type is displayed: - Contact Person - Co-Applicant - Roommate - Contact Person
3	Enter zeros to SSN/FEIN field Enter 1 number to SSN/FEIN field Enter more than 9 numbers to SSN/FEIN field Enter alphabetical and special symbols to SSN/FEIN field	The user can't start from "0". The keyboard is locked. The "Please enter correct SSN/FEIN number (9 numeric chars)" message is displayed The user can't enter more than 9 symbols. The keyboard is locked. The user can't enter alphabetical and special symbols. The keyboard is locked.

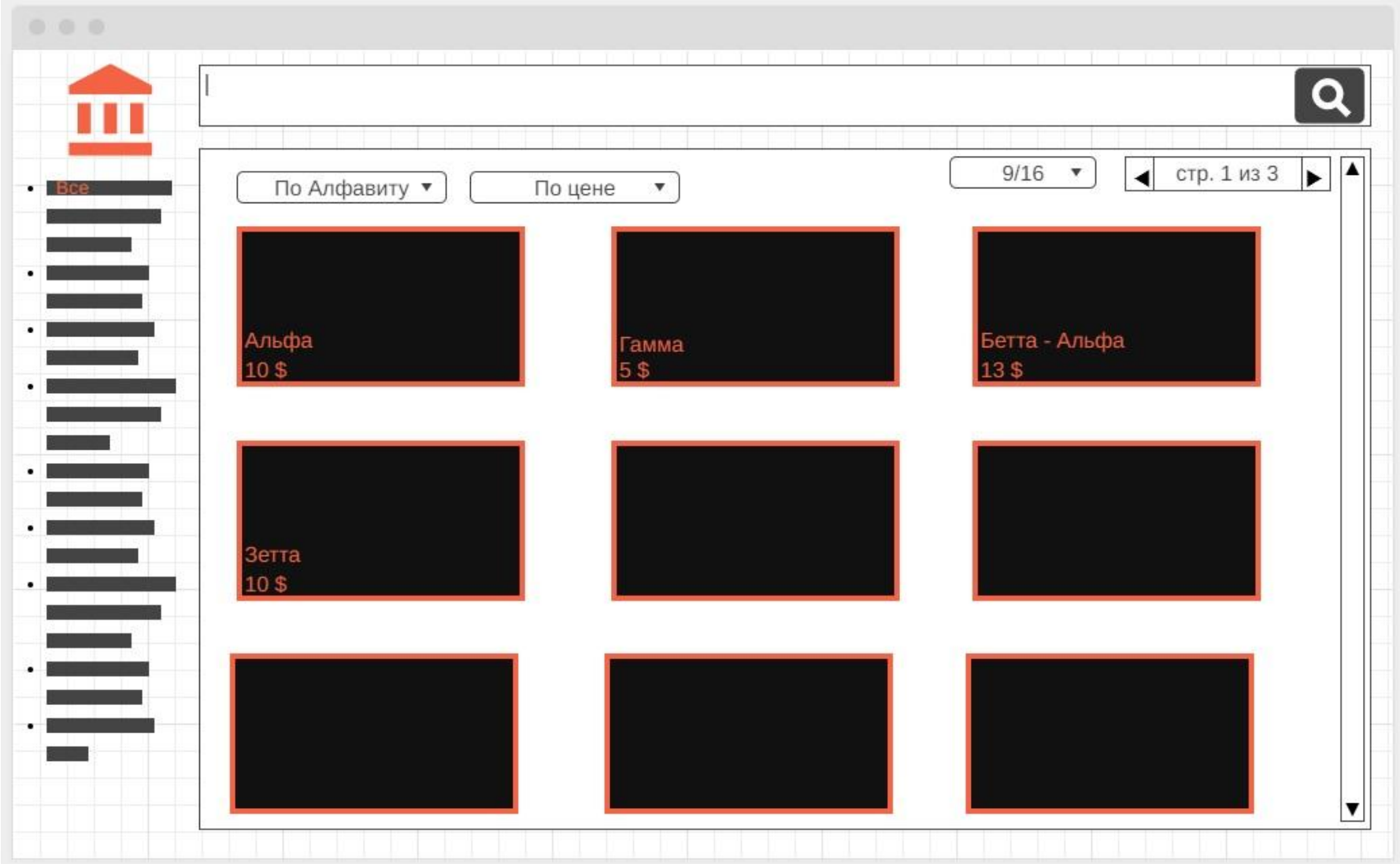
1) Список Каталогов:
- Каталоги отображаются по алфавиту
- Каталог "Все" отображается первым
- При выборе каталога - отображаются товары входящие в него

2) Список товаров:
- Отображается 9 товаров по умолчанию
- Сортировка по алфавиту, по возрастанию по умолчанию

3) Сортировка:
- По алфавиту. По возрастани, По убыванию
- По цене. По возрастани, По убыванию
- Можно выбрать только один вариант
- При выборе сортировки отображается знак сортировки (треугольник)

4) Пейджинация:
- Можно выбрать отображение 9 или 16 товаров на страницу
- Можно переключать страницы
- Отображается номер страницы и общее число страниц

5) Поиск:
- Можно искать по слову целиком
- Можно искать по части слова
- Поиск происходит по нажатию иконки поиска. И по нажатию клавиши "Enter" на клавиатуре



Формат кода

Практика. Написание Отчёта о дефекте

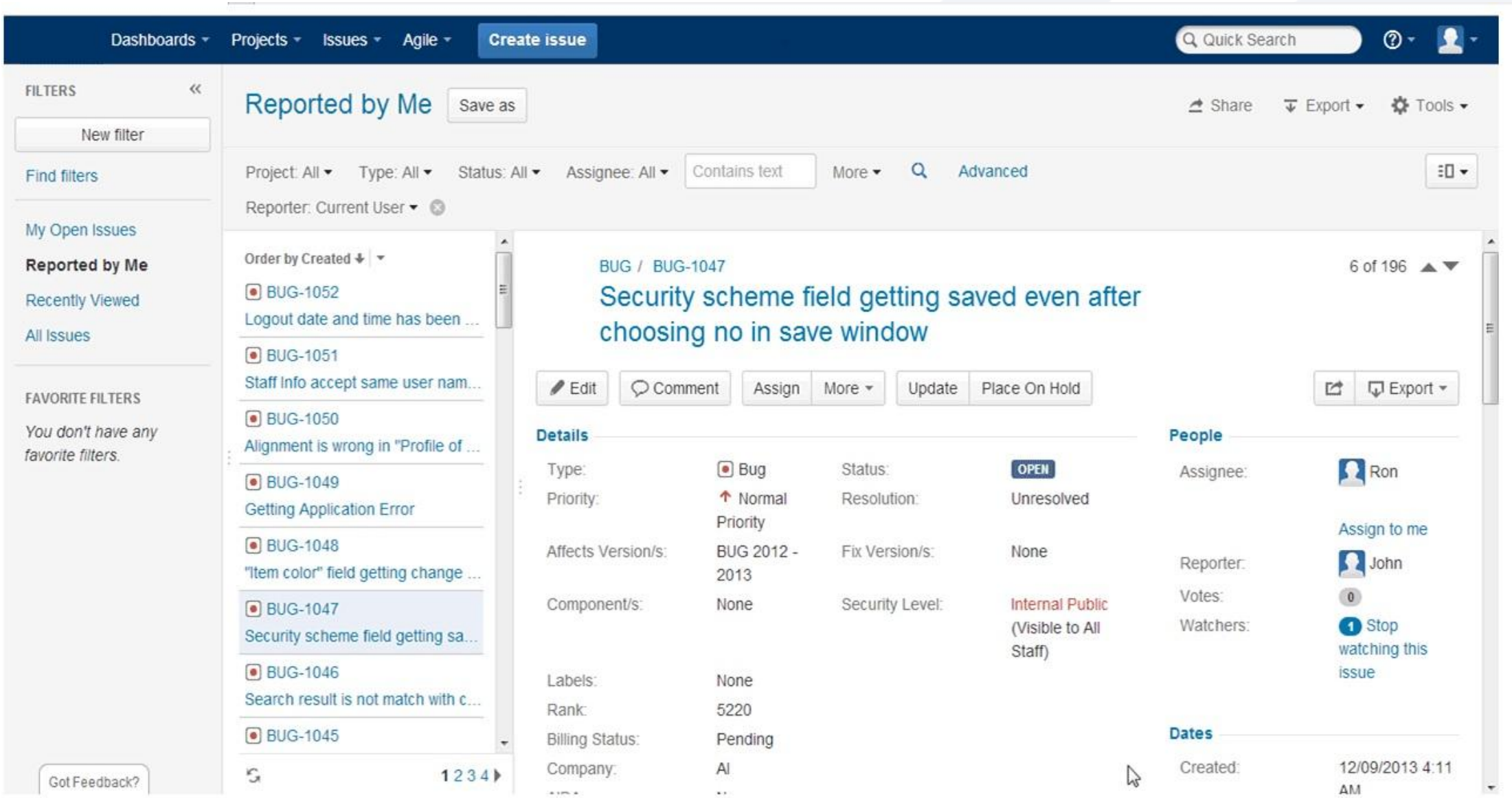
Отчёт о дефекте (bug report) — это документ, описывающий ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

Атрибуты отчёта о дефекте:

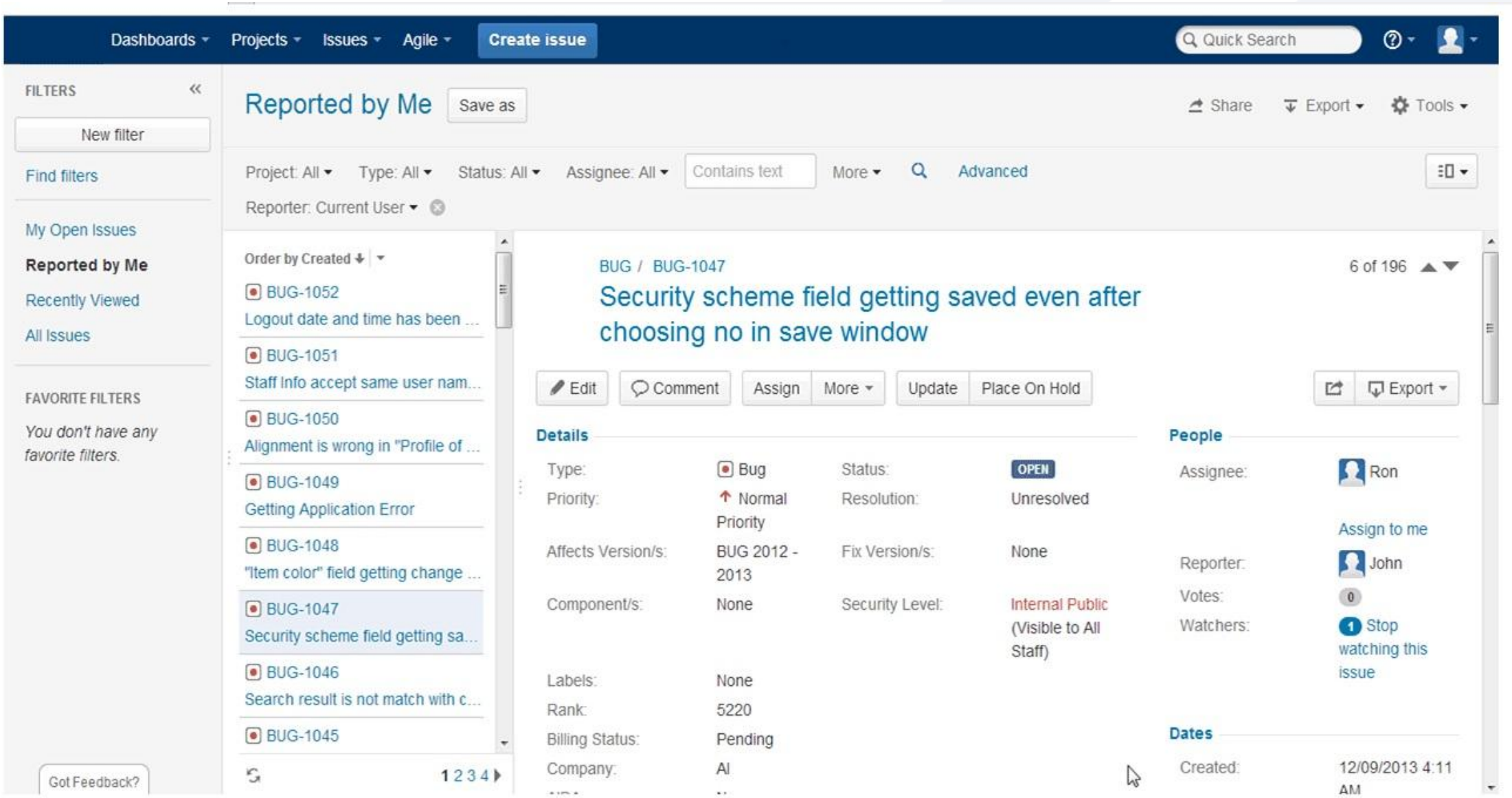
- **Уникальный идентификатор (ID)**
- **Имя (Тема, краткое описание, Summary)**
- **Подробное описание (Description)**
- **Шаги для воспроизведения (Steps To Reproduce)**
- **Фактический результат (Actual result)**
- **Ожидаемый результат (Expected result)**
- **Вложения (Attachments)**
- **Серьёзность дефекта (важность, Severity)**
- **Приоритет дефекта (срочность, Priority)**
- **Статус (Status)**

Формат <кода>

Практика. Написание Отчёта о дефекте



Практика. Написание Отчёта о дефекте



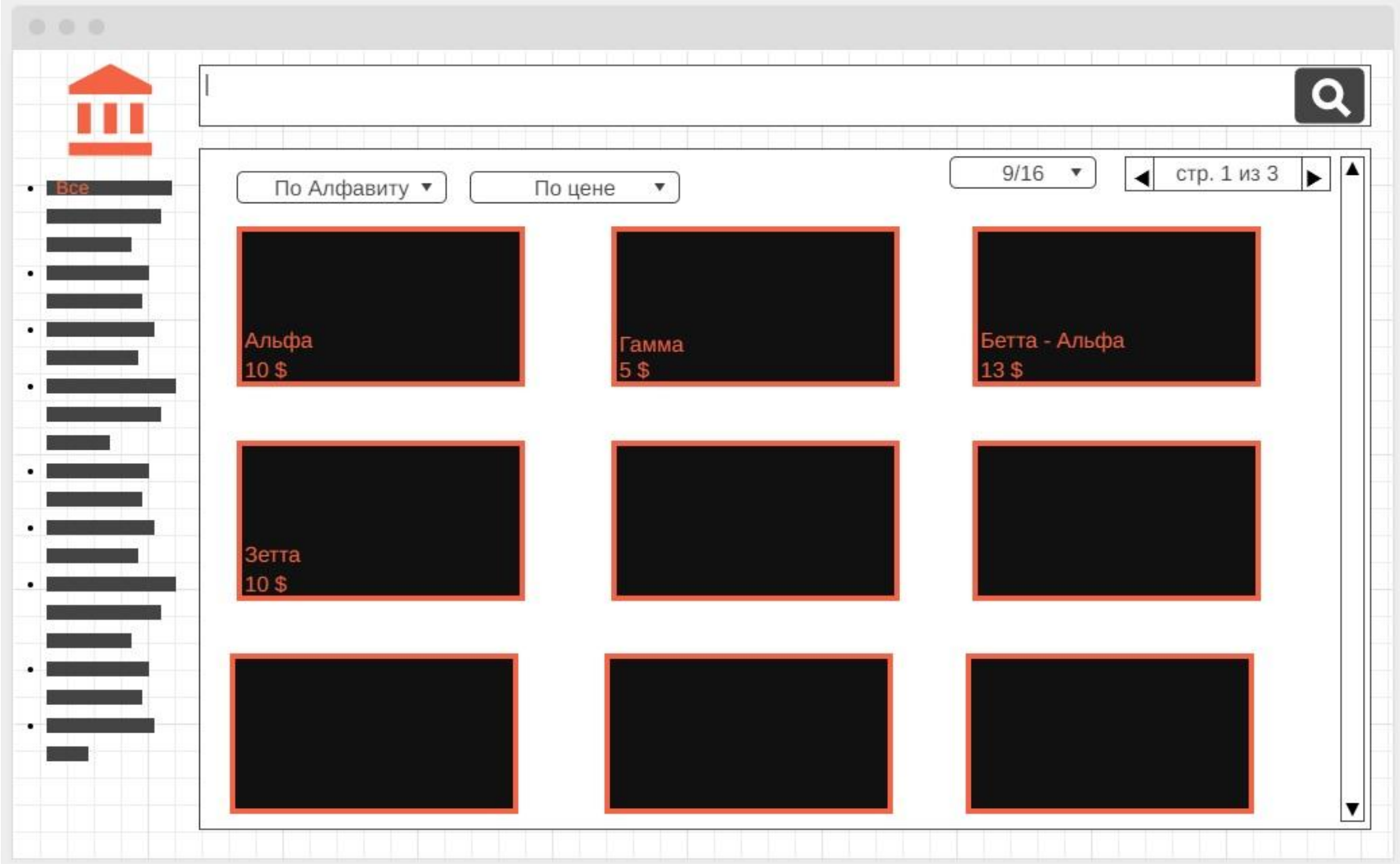
1) Список Каталогов:
- Каталоги отображаются по алфавиту
- Каталог "Все" отображается первым
- При выборе каталога - отображаются товары входящие в него

2) Список товаров:
- Отображается 9 товаров по умолчанию
- Сортировка по алфавиту, по возрастанию по умолчанию

3) Сортировка:
- По алфавиту. По возрастани, По убыванию
- По цене. По возрастани, По убыванию
- Можно выбрать только один вариант
- При выборе сортировки отображается знак сортировки (треугольник)

4) Пейджинация:
- Можно выбрать отображение 9 или 16 товаров на страницу
- Можно переключать страницы
- Отображается номер страницы и общее число страниц

5) Поиск:
- Можно искать по слову целиком
- Можно искать по части слова
- Поиск происходит по нажатию иконки поиска. И по нажатию клавиши "Enter" на клавиатуре



Формат кода

Проверочная работа

Проверочная работа

1) Что такое «тестирование»?
2) Перечислить атрибуты Тест-Кейса
3) Перечислить атрибуты Отчёта Об Дефекте
4) Написать Чек-Лис
5) Написать 1-2 Тест-Кейса

Формат <кода>

Проверочная работа

1) Что такое «тестирование»?
2) Перечислить атрибуты Тест-Кейса
3) Перечислить атрибуты Отчёта Об Дефекте
4) Написать Чек-Лис
5) Написать 1-2 Тест-Кейса

Требования к Форме Входа:

1) Поле «Логин»:
   a) Длинна от 3 до 6 символов включительно
   b) Не чувствительно к регистру
2) Поле «Пароль»
   a) Длинна от 5 до 10 символов включительно
   b) Чувствительно к регистру
3) Кнопка «Войти»
   a) Не активна до ввода «Логина» и «Пароля»
   b) Можно войти по клику или нажатию кнопки «Enter» на клавиатуре
   c) При успешном входе открывается страницы личного кабинета
   d) При не успешном входе – выводится сообщение «Неверный Логин или Пароль»

Формат <кода>

Формат <кода>

- https://playground.learnqa.ru/puzzle/triangle

Формат <кода>

ПРИМЕРЫ ПРОЕКТОВ

Название раздела

ПРИМЕРЫ ПРОЕКТОВ

Название раздела

Спасибо!

formatkoda.ru

<имя>
<email>
<phone#/other contacts>