1. **История тестирования программного обеспечения**

Первые программные системы разрабатывались в рамках программ научных исследований или программ для нужд министерств обороны.

Тестирование таких продуктов проводилось строго формализовано с записью всех тестовых процедур, тестовых данных, полученных результатов.

Тестирование выделялось в отдельный процесс, который начинался после завершения кодирования, но при этом, как правило, выполнялось тем же персоналом.

**В 1960-х много** внимания уделялось «исчерпывающему» тестированию, которое должно проводиться с использованием всех путей в коде или всех возможных входных данных.

Однако это невозможно:

#1 количество возможных входных данных очень велико;

#2 существует множество путей;

#3 сложно найти проблемы в архитектуре и спецификациях.

Итог: «исчерпывающее» тестирование было отклонено и признано теоретически невозможным.

**В начале 1970-х годов** тестирование программного обеспечения обозначалось как «процесс, направленный на демонстрацию корректности продукта» или как «деятельность по подтверждению правильности работы программного обеспечения».

Впоследствии этот метод тестирования был признан неэффективным.

**Во второй половине 1970-х** тестирование представлялось как выполнение программы с намерением найти ошибки, а не доказать, что она работает.

Успешный тест — это тест, который обнаруживает ранее неизвестные проблемы.

**В 1980-е годы** тестирование расширилось таким понятием, как предупреждение дефектов.

Стали высказываться мысли, что необходима методология тестирования, в частности, что тестирование должно включать проверки на всем протяжении цикла разработки, и это должен быть управляемый процесс.

В ходе тестирования надо проверить не только собранную программу, но и требования, код, архитектуру, сами тесты.

**В начале 1990-х годов** в понятие «тестирование» стали включать планирование, проектирование, создание, поддержку и выполнение тестов и тестовых окружений, и это означало переход от тестирования к обеспечению качества.

Начинают появляться различные программные инструменты для поддержки процесса тестирования: более продвинутые среды для автоматизации с возможностью создания скриптов и генерации отчетов, системы управления тестами, ПО для проведения нагрузочного тестирования.

**С нулевых годов по настоящее время** появляются и получают широкое распространение гибкие методологии, новые подходы. Широко используется автоматизация.

У тестировщиков появляются специализированные технологии и инструменты, а сам процесс тестирования интегрирован в цикл разработки программного обеспечения.

1. **Обеспечение качества: терминология, ISO/IEC 25010:2011 (Модель качества при использовании)**

**Обеспечение качества** **(Quality Assurance - QA) - это** совокупность мероприятий, охватывающих все технологические этапы разработки, выпуска и эксплуатации программного обеспечения информационных систем, предпринимаемых на разных стадиях жизненного цикла ПО, для обеспечения требуемого уровня качества выпускаемого продукта.

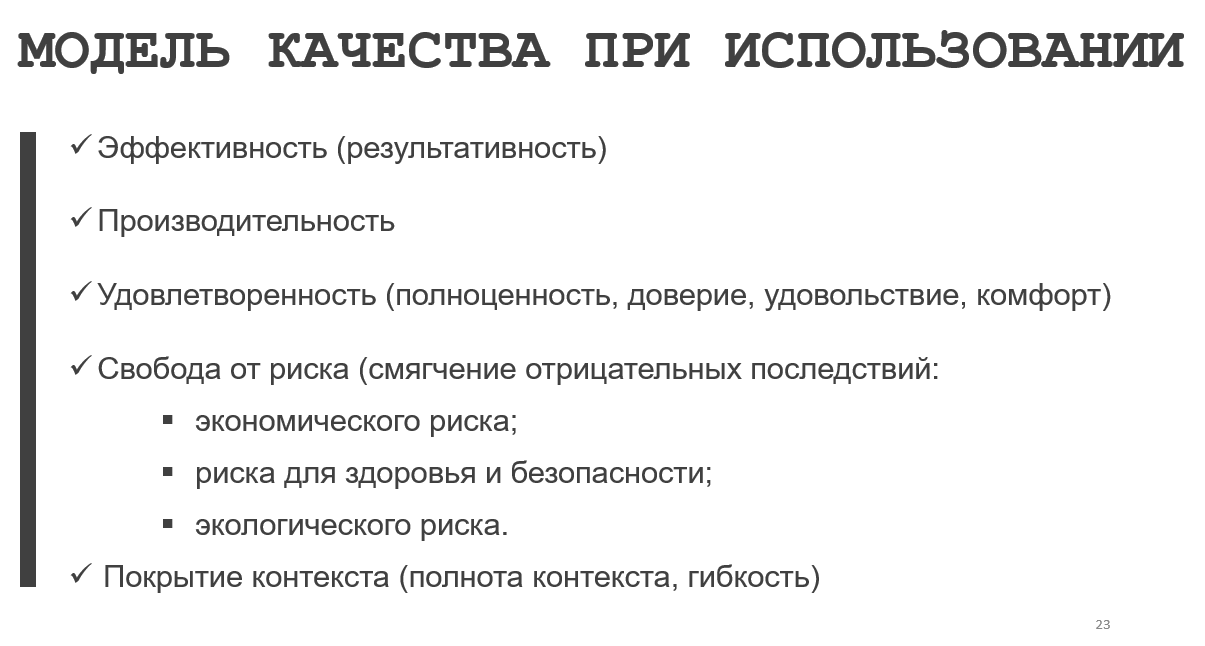
**Контроль качества** **(Quality Control - QC) - это** совокупность действий, проводимых над продуктом в процессе разработки, для получения информации о его актуальном состоянии в разрезах: "готовность продукта к выпуску", "соответствие зафиксированным требованиям", "соответствие заявленному уровню качества продукта".

**Тестирование программного обеспечения (Software Testing)** **- это** одна из техник контроля качества, включающая в себя активности по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).

Качество системы – это степень удовлетворения системой заявленных и подразумеваемых потребностей различных заинтересованных сторон, которая позволяет, таким образом, оценить достоинства.

Модели качества:

* Модель качества при использовании
* Модель качества продукта
* Модель качества данных ISO/IEC 25012:2008



4.1.1 **эффективность, результативность** (effectiveness): Точность и полнота, с которой пользователи достигают определенных целей.

4.1.2 **эффективность, производительность** (efficiency): Связь точности и полноты достижения пользователями целей с израсходованными ресурсами.

4.1.3 **удовлетворенность** (satisfaction): Способность продукта или системы удовлетворить требованиям пользователя в заданном контексте использования.

4.1.3.1 **полноценность** (usefulness): Степень удовлетворенности пользователя достижением прагматических целей, включая результаты использования и последствия использования.

4.1.3.2 **доверие** (trust): Степень уверенности пользователя или другого заинтересованного лица в том, что продукт или система будут выполнять свои функции так, как это предполагалось.

4.1.3.3 **удовольствие** (pleasure): Степень удовольствия пользователя от удовлетворения персональных требований.

4.1.3.4 **комфорт** (comfort): Степень удовлетворенности пользователя физическим комфортом.

4.1.4 **свобода от риска** (freedom from risk): Способность продукта или системы смягчать потенциальный риск для экономического положения, жизни, здоровья или окружающей среды.

4.1.4.1 **смягчение отрицательных последствий экономического риска** (economic risk mitigation): Способность продукта или системы смягчать потенциальный риск для финансового положения и эффективной работы, коммерческой недвижимости, репутации или других ресурсов в предполагаемых условиях использования.

4.1.4.2 **смягчение отрицательных последствий риска для здоровья и безопасности** (health and safety risk mitigation): Способность продукта или системы смягчать потенциальный риск для людей в предполагаемых условиях использования.

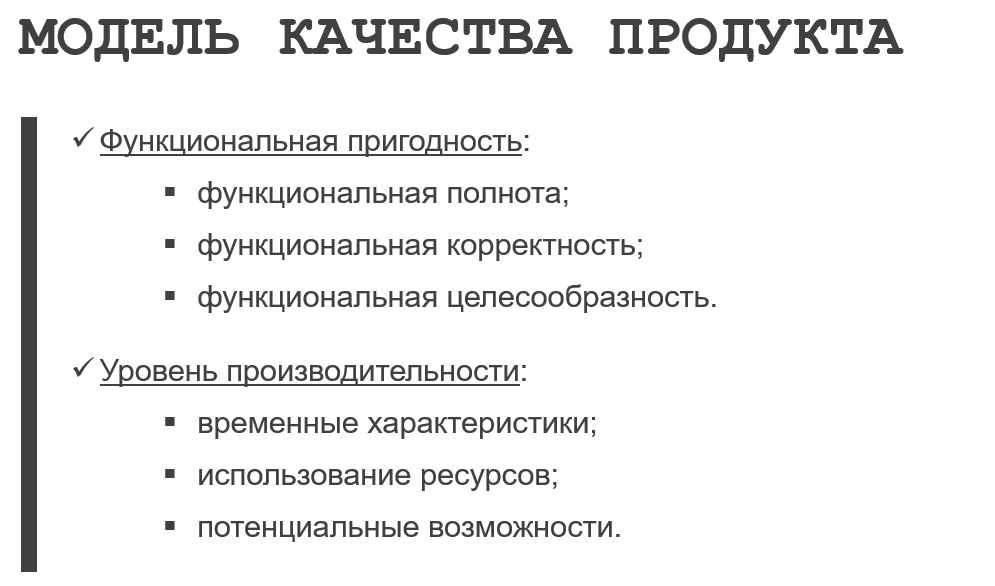
4.1.4.3 **смягчение отрицательных последствий экологического риска** (environmental risk mitigation): Способность продукта или системы смягчать потенциальный риск для имущества или окружающей среды в предполагаемых условиях использования.

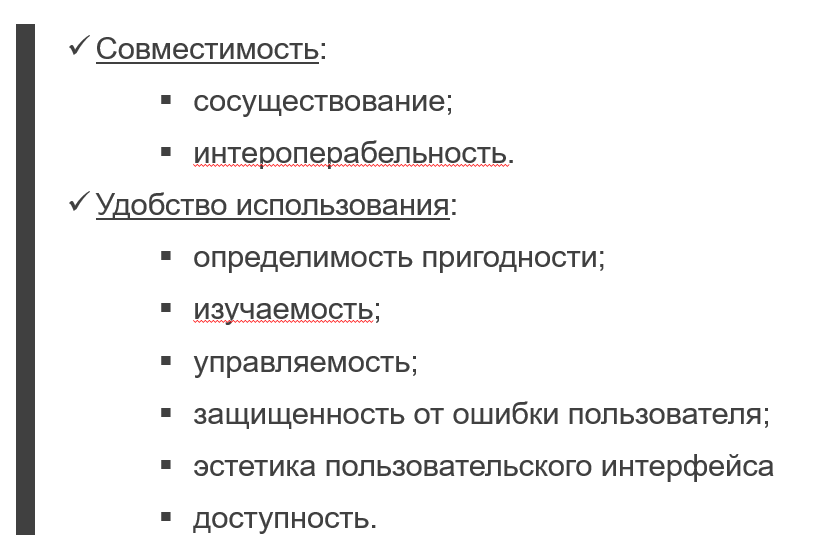
4.1.5 **покрытие контекста** (context coverage): Степень, в которой продукт или система могут быть использованы с эффективностью, результативностью, свободой от риска и в соответствии с требованиями как в первоначально определенных условиях использования, так и в условиях, выходящих за спецификации.

4.1.5.1 **полнота контекста** (context completeness): Степень, в которой продукт или система могут быть использованы с эффективностью, результативностью, свободой от риска и в соответствии с требованиями при всех указанных условиях использования.

4.1.5.2 **гибкость** (flexibility): Степень, в которой продукт или система могут быть использованы с эффективностью, результативностью, свободой от риска и в соответствии с требованиями в условиях, выходящих за рамки первоначально определенных в требованиях.

1. **Модель качества продукта по стандарту ISO/IEC 25010:2011**





4.2.1 **функциональная пригодность** (functional suitability): Степень, в которой продукт или система обеспечивают выполнение функции в соответствии с заявленными и подразумеваемыми потребностями при использовании в указанных условиях.

4.2.1.1 **функциональная полнота** (functional completeness): Степень покрытия совокупностью функций всех определенных задач и целей пользователя.

4.2.1.2 **функциональная корректность** (functional correctness): Степень обеспечения продуктом или системой необходимой степени точности корректных результатов.

4.2.1.3 **функциональная целесообразность** (functional appropriateness): Степень функционального упрощения выполнения определенных задач и достижения целей.

4.2.2 **уровень производительности** (performance efficiency): Производительность относительно суммы использованных при определенных условиях ресурсов.

4.2.2.1 **временные характеристики** (time behaviour): Степень соответствия требованиям по времени отклика, времени обработки и показателей пропускной способности продукта или системы.

4.2.2.2 **использование ресурсов** (resource utilization): Степень удовлетворения требований по потреблению объемов и видов ресурсов продуктом или системой при выполнении их функций.

4.2.2.3 **потенциальные возможности** (capacity): Степень соответствия требованиям предельных значений параметров продукта или системы.

4.2.3 **совместимость** (compatibility): Способность продукта, системы или компонента обмениваться информацией с другими продуктами, системами или компонентами, и/или выполнять требуемые функции при совместном использовании одних и тех же аппаратных средств или программной среды.

4.2.3.1 **сосуществование (совместимость)** (co-existence): Способность продукта совместно функционировать с другими независимыми продуктами в общей среде с разделением общих ресурсов и без отрицательного влияния на любой другой продукт.

4.2.3.2 **функциональная совместимость** (интероперабельность) (interoperability): Способность двух или более систем, продуктов или компонент обмениваться информацией и использовать такую информацию.

4.2.4 **удобство использования** (usability): Степень, в которой продукт или система могут быть использованы определенными пользователями для достижения конкретных целей с эффективностью, результативностью и удовлетворенностью в заданном контексте использования.

4.2.4.1 **определимость пригодности** (appropriateness recognizability): Возможность пользователей понять, подходит ли продукт или система для их потребностей, сравним ли с функциональной целесообразностью (functional appropriateness).

4.2.4.2 **изучаемость** (learnability): Возможность использования продукта или системы определенными пользователями для достижения конкретных целей обучения для эксплуатации продукта или системы с эффективностью, результативностью, свободой от риска и в соответствии с требованиями в указанном контексте использования.

4.2.4.3 **управляемость** (operability): Наличие в продукте или системе атрибутов, обеспечивающих простое управление и контроль.

4.2.4.4 **защищенность от ошибки пользователя** (user error protection): Уровень системной защиты пользователей от ошибок.

4.2.4.5 **эстетика пользовательского интерфейса** (user interface aesthetics): Степень "приятности" и "удовлетворенности" пользователя интерфейсом взаимодействия с пользователем.

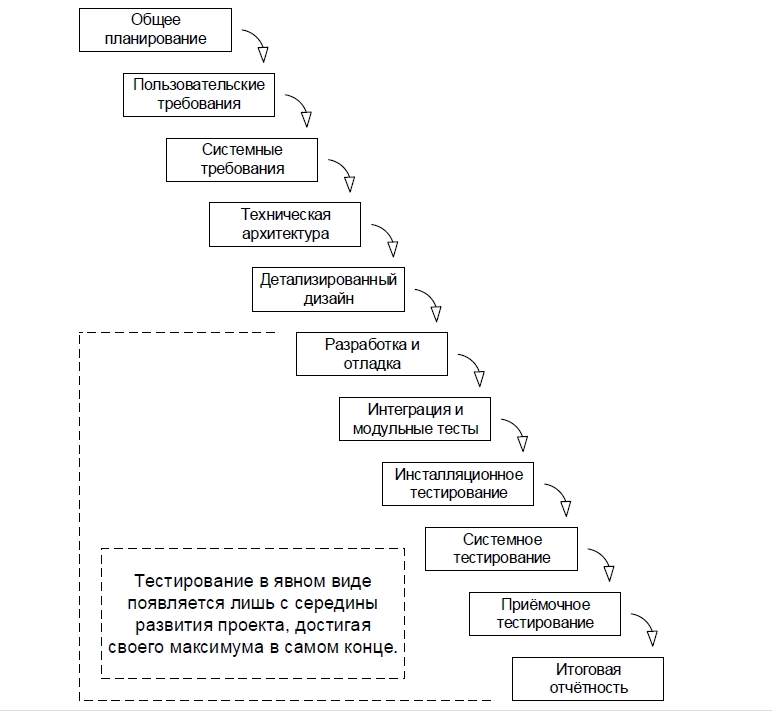
4.2.4.6 **доступность** (accessibility): Возможность использования продукта или системы для достижения определенной цели в указанном контексте использования широким кругом людей с самыми разными возможностями.

1. **Процесс и методологии разработки программного обеспечения (Водопадная, V-образная, Итерационная)**

Модель разработки ПО – структура, систематизирующая различные виды проектной деятельности, их взаимодействие и последовательность в процессе разработки ПО. Выбор той или иной модели зависит от масштаба и сложности проекта, предметной области, доступных ресурсов и множества других факторов.

Жизненный цикл ПО – период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

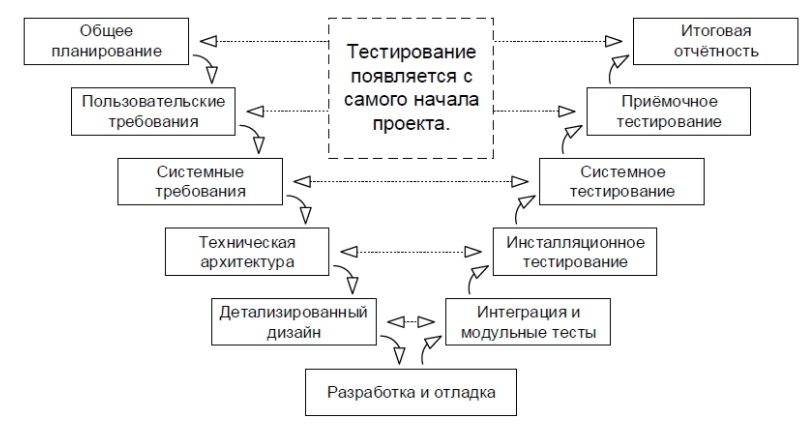
**Водопадная модель:**



Waterfall не подходит, если часто меняются требования и для маленькой команды, применяется для большой команды и в разработке ПО для медицины или военных. Из минусов – ошибки стоят очень дорого и могут жить в проекте очень долго.

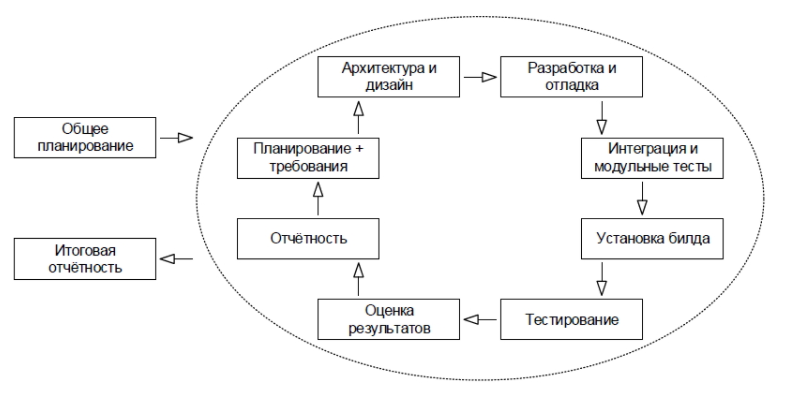
**V-образная модель:**

V-образная модель представляет собой итерационную разновидность каскадной модели и предполагает уже на ранних этапах жизненного цикла планирование работ по тестированию программного продукта. Так, например, уже на этапе разработки требований к проекту намечается план проведения заказчиком приемо-сдаточных испытаний.

****

Подходит на небольшие проекты.

**Итерационная модель:**

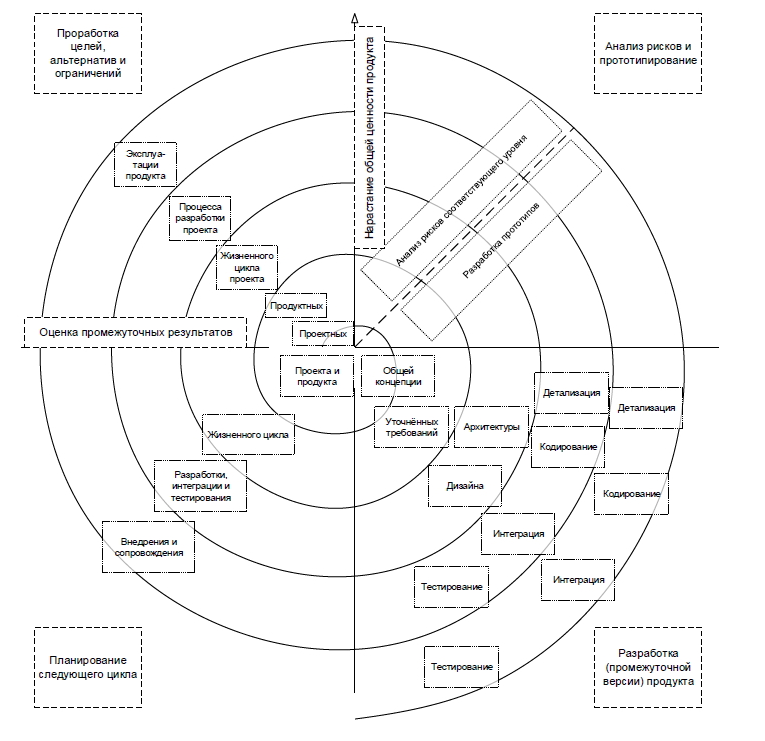


Итеративная (итерационная) модель предполагает движение к выбранному финальному варианту продукта через повторяющиеся циклы разработки. Такие циклы называются итерациями. После каждого цикла создается новая версия ПО. По мере продвижения по итерациям IT-продукт становится все более качественным и удобным.

Подходит для небольших проектов и команд, не подходит для медицинского и военного ПО.

1. **Процесс и методологии разработки программного обеспечения (Спиральная, Гибкая, SCRUM)**

**Спиральная модель:**



Каждый цикл является фазой и их количество варьируется между проектами и неизвестно вначале. Радиус спирали – стоимость проекта на данном этапе. Помогает справляться с рисками на каждом этапе, так как составляется прототип и в будущем дополняется. Подходит бля больших проектов, является очень затратным вариантом.

**Гибкая модель:**

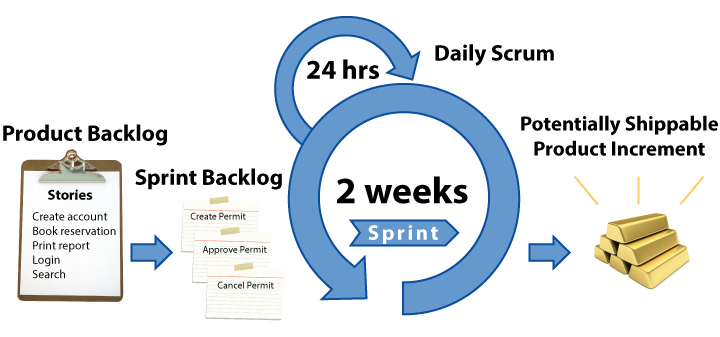
Agile даже не модель, а философия разработки, которая имеет собственный манифест состоящий из 4 пунктов (ценности):

* **Люди и взаимодействие** важнее процессов и инструментов.
* **Работающий продукт** важнее исчерпывающей документации.
* **Сотрудничество с заказчиком** важнее согласования условий контракта.
* **Готовность к изменениям** важнее следования первоначальному плану.

Также есть 11 принципов основанных на этих четырех ценностях.

Эту философию реализуют следующие методологии: Scrum, Kanban. XP, RUP, MSF, DSDM и тд.

**SCRUM модель:**



Product Backlog – то, что хочу сделать или должно быть реализовано в продукте (набор стори (историй, функций)).

Sprint Backlog – то, что будет реализовано за спринт.

Sprint – период, в котором реализуются функции из Sprint Backlog от 1 до 6 недель.

В конце получается готовый продукт (новая версия), который можно использовать.

Product Owner – владелец продукта, заказчик, делает истории для Product Backlog, выставляет приоритеты.

Scrum Master – отвечает за процесс Scrum, следит за всем, является одним из разработчиков.

Development team – команда разработчиков.

Sprint Grooming – мероприятие на котором определяется, что будет реализовано и каким образом.

Sprint Planning – делается в первый день спринта, создание sprint backlog и разделение задач.

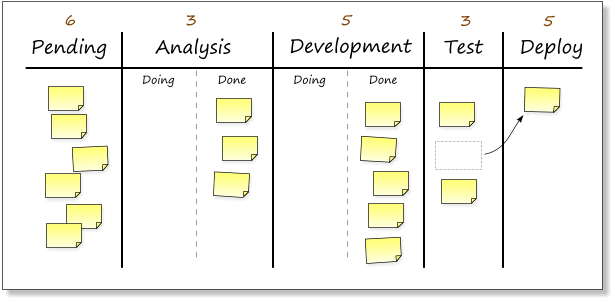
Daily standup (scrum) – 10-15 минутный переговор, обсуждают что сделал вчера и что будет делать сегодня и обсуждают проблемы.

Sprint Review – в конце спринта, что было сделано, получение фидбека.

Sprint Retrospective – что было хорошо в спринте, а что плохо.

1. **Процесс и методологии разработки программного обеспечения (Канбан, Экстремальное программирование)**

**Kanban:**



Основной элемент Канбана – доска (числа над столбцами означают количество элементов в ней).

* В Kanban нет таймбоксов
* В Kanban задачи объемнее, но их меньше
* В Kanban оценки сроков на задачу опциональные или вообще их нет
* В Kanban «скорость работы команды» отсутствует и считается только

среднее время на полную реализацию задачи

**Экстремальное программирование - методология:**



Часто делаются маленькие кусочки, сначала делается простой дизайн и архитектура, которая в последствии усложняются. Иногда применяется парное программирование. 40-часовая неделя с редкими переработками.

Практики реализующие XP: TDD, BDD.

1. **Процесс тестирования программного обеспечения (Планирование и управление тестированием, Анализ и проектирование тестов, Внедрение и реализация тестов, Оценка критериев выхода и создание отчетов)**

Планирование тестирования – это действия, направленные на определение целей тестирования и описание задач тестирования для достижения этих целей и миссии (анализ требований, определение целей тестирования, определение общего подхода к тестированию, составление графика).

Управление тестированием – это постоянное сопоставление текущего положения дел с планом и отчетность о состоянии дел, включая отклонения от плана.

Анализ и проектирование тестов – это деятельность, во время которой общие цели тестирования материализуются в тестовые условия и тестовые сценарии (расстановка приоритетов тестирования, выявление необходимых данных для поддержки тестовых условий и тестовых сценариев, выявление необходимых инструментов).

Внедрение и реализация тестов – это деятельность, где процедуры тестирования или автоматизированные сценарии задаются последовательностью тестовых сценариев, а также собирается любая информация, необходимая для выполнения тестов, разворачивается окружающая среда, и запускаются тесты (завершение, реализация и расстановка приоритетов тестовых сценариев, написание автоматизированных сценариев тестирования, проверка правильности настройки тестового окружения, выполнение процедур тестирования либо вручную, либо используя инструменты выполнения тестов, согласно заданному плану, регистрация результатов выполнения тестов, сравнение фактических и ожидаемых результатов).

Критерии выхода и отчетность – это деятельность, где выполнение тестов оценивается согласно определенным целям. Она должна быть выполнена для каждого уровня тестирования (анализ необходимости использования дополнительных тестов или изменения критериев выхода, стоимость, степень покрытия кода, оценка плотности дефектов).

Действия по завершению тестов – это сбор данных о завершенных испытаниях для объединения опыта, тестового обеспечения, фактов и цифр. Проводится после релиза (проверка, что запланированные результаты достигнуты, передача тестового обеспечения организации сопровождения, завершение и архивирование тестового обеспечения, тестового окружения и инфраструктуры тестирования для последующего использования).

1. **Уровни тестирования программного обеспечения, Верификация и валидация**

**Уровни тестирования ПО:**

* Компонентное тестирование (new feature testing vs unit testing) – тип тестирования ПО, при котором тестирование выполняется для каждого отдельного компонента отдельно, без интеграции с другими компонентами.
* Интеграционное тестирование – это тип тестирования, при котором программные модули объединяются логически и тестируются как группа. Как правило, программный продукт состоит из нескольких программных модулей, написанных разными программистами. Целью нашего тестирования является выявление багов при взаимодействии между этими программными модулями и в первую очередь направлен на проверку обмена данными между этими самими модулями.
* Системное тестирование – это вид тестирования программного обеспечения, который выполняет проверку системы в целом. Она включает в себя интеграцию всех отдельных модулей и компонентов разработанного вами программного обеспечения, чтобы проверить, работает ли система вместе, как ожидалось.
* Приемочное тестирование – формальное тестирование по отношению к потребностям, требованиям и бизнес процессам пользователя, проводимое с целью определения соответствия системы критериям приемки и дать возможность пользователям, заказчикам или иным авторизированым лицам определить, принимать систему или нет.

Верификация – это процесс тестирования приложения на соответствие предъявленным требованиям осуществляется на протяжении всей разработки приложения (компонентное, интеграционное, системное тестирование).

Валидация – это проверка приложения на соответствие бизнес-целям проводится, как правило, перед релизом (приемочное тестирование).

1. **Типы тестирования программного обеспечения (Статическое и динамическое, Ручное и автоматизированное)**

**Метод статического тестирования** – это тип тестирования ПО, где программное обеспечение проверяется без запуска кода; является процессом или инструментом, направленным на обнаружение возможных багов в ПО. Кроме этого, он находит и устраняет ошибки в разного рода сопроводительных документах, например, специфики требований к ПО (вычитка исходного кода, проверка требований).

**Динамическое тестирование** – это методика, направленная на проверку функционала программы, во время выполнения кода. То есть, данный тип тестирования подразумевает фактическую эксплуатацию программы и определение того, как работает ее функционал, в соответствии с ожиданиями или нет (компонентное, интеграционное и приемочное тестирование).

**Ручное тестирование** – это вид тестирования программного обеспечения, при котором тесты выполняются тестировщиком вручную, без использования каких-либо средств автоматизации.

Плюсы: отчет тестировщика, обратная связь по UI, низкая стоимость, возможность тестировать нетипичные ситуации.

Минусы: человеческий фактор, долгая продолжительность.

**Автоматизированное тестирование** – это метод тестирования программного обеспечения, который предполагает использование инструментов и фреймворков автоматизации для выполнения одного и того же набора тест-кейсов снова и снова.

Плюсы: скорость работы, отсутствие человеческого фактора, многократное использование, работа с большими объемами данных.

Минусы: необходимость в квалифицированном персонале, отсутствие обратной связи, невозможность протестировать UI, невозможность взглянуть от лица пользователя.

1. **Методы тестирования программного обеспечения (Белого, черного, серого ящиков)**

**Белый ящик** – тестирование исходного кода ПО.

Плюсы: упрощение диагностики, легко автоматизировать тесты, легко собирать данные.

Минусы: высокий порог входа, не учитывается непредсказуемость пользователей.

**Черный ящик** – тестирование ПО через интерфейс (без доступа к исходному коду).

Плюсы: низкий порог входа, учитывается среда выполнения, тестирование требований.

Минусы: вероятность неполного покрытия, необходимость качественной документации.

**Серый ящик** – комбинация белого и черного ящика.

Плюсы: возможность усложнения тестов, легко убрать ненужные тесты.

Минусы: ограничен анализ кода.

1. **Виды тестирования программного обеспечения (функциональное, нефункциональное, структурное, тестирование изменений, тестирование по приоритетам)**

**Функциональное тестирование** – тестирование, которое разрабатывается на основе функций и возможностей системы и их взаимодействия со специфичными системами и могут быть выполнены на всех уровнях тестирования. Проводится методом «черного ящика».

Примеры: позитивное, негативное, CRUD и тестирование по сценариям использования.

**Нефункциональное тестирование** – тестирование, которое проводится для оценки характеристик систем и программ. Проверяется не корректность работы функций приложения, а сопутствующие характеристики.

Примеры: тестирование внешнего вида приложения (методом «черного ящика»), нагрузочное тестирование (методом «белого ящика»), тестирование безопасности (методом «черного ящика»), тестирование совместимости (методом «черного ящика»).

**Структурное тестирование** – анализ и тестирование кода продукта, его архитектуры. Проводится методом «белого ящика».

Примеры: Unit-тесты, интеграционные автоматизированные тесты, тестирование веб-сервисов.

**Тестирование изменений** – это повторное тестирование уже протестированных программ после внесения в них изменений, чтобы обнаружить дефекты, внесенные или пропущенные в результате этих действий. Чаще проводится методом «черного ящика».

Примеры: регрессионное тестирование, тестирование основанное на рисках.

**Тестирование по приоритету** – виды тестирования, направленные на выявление качества функционала определенной важности.

Примеры: дымовое тестирование (smoke test), тестирование критического пути (critical path test), расширенное тестирование (extended test).

Smoke test – тестирование, направленное на определение на проверку самой главной, самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования фичи или приложения.

Critical path test – тестирование, направленное на исследование функциональности, используемой типичными пользователями в типичной повседневной деятельности.

Extended test – тестирование, направленное на исследование всей заявленной в требованиях функциональности — даже той, которая имеет низкий приоритет.

1. **Функциональное тестирование, Модели поведения пользователя, Позитивное и негативное тестирование**

Базируется на функциях и особенностях, а также на взаимодействии с другими системами.

Проводится в двух аспектах: требования и бизнес-процессы.

Этапы: подготовка, проведение, отчет.

Виды: позитивное, негативное, исследовательское, интуитивное, CRUD, тестирование по сценариям использования, тестирование основанное на ролях.

**Позитивное тестирование** – тестирование, при котором используются только валидные данные и выполняются только валидные действия.

**Негативное тестирование** – тестирование с использованием невалидных данных и действий, направленное на получение ошибок и предупреждений.

**Модели поведения пользователя:**

**Пользователь-интуит** – пользователь не читал инструкций или не способен их прочитать. Находится несоответствие интерфейса программы существующим стереотипам.

**«Хороший» пользователь** –добросовестный пользователь действует в строгом соответствии с инструкциями. Поиск ошибок как в логике работы программы, так и в документации на программу.

**«Плохой» пользователь** **–** недобросовестный пользователь стремится использовать программу непредусмотренным способом.

1. **Исследовательское тестирование ПО (что это, отличие от сценарного тестирования, когда стоит и когда не стоит применять)**

**Исследовательское тестирование** – подход к тестированию, который подразумевает под собой одновременно изучение приложения, проектирование тестовых сценариев и их немедленное выполнение.

Является противоположностью сценарного тестирование (при исследовательском используется интуитивный подход, то есть мы изучаем приложение), при сценарном все делает на основе сценариев по use-кейсам.

Применяются когда осталось мало времени на тестирование, есть сложности с требованиями, небольшой проект.

Следует избегать когда приложение стандартизировано, тестирование отдается на аутсорс, длительный проект.

Цели: проверить работу системы с точки зрения конечного пользователя, найти баги, которые не были найдены другими способами, изучить работу приложения без опоры на документацию.

1. **Исследовательское тестирование ПО (Туры, Туры по бизнес-центру)**

Туры – это идеи и инструкции по исследованию программного продукта, объединенные определённой общей темой или целью. Туры, как правило, ограничены по времени – длительность тестовой сессии не должна превышать 4 часа.

Бизес-центр – Это места, где «делается бизнес». Они начинаются с запуска кода и до завершения его работы. В них расположены функции, ради которых пользователи и используют данное приложение. Это «задняя стенка коробки» (читай – приложения), которая является главной при демонстрации коммерческой ценности продукта, а также код, который это поддерживает.

Экскурсии по путеводителю – при таком подходе приоритет отдается основным функциям приложения, что позволяет точно воспроизвести работу обычного пользователя с программой и выявить проблемы, которые он обнаружит естественным образом.

**Денежный тур – э**тот исследовательский тест проверяет критически важные функции приложения, в частности те, за доступ к которым клиенты и заказчики платят деньги — как правило, они являются наиболее приоритетными для команды тестирования.

Путешествие по ориентирам – мы выбираем какие-либо ориентиры и осуществляем скачок к одному из них через все приложение. Например, мы выбираем ключевую функцию в качестве отметки, которую мы определили в туре по путеводителю или в денежном туре. Далее мы составляем набор из таких отметок, определяем их порядок и исследуем приложение от отметки к отметке, пока не достигнем каждой из них. Далее можно создать карту покрытия отметками. Разный порядок, разные отметки – вот поле для вариативности подобного тура.

**Интеллектуальный тур –** Этот тур продвигает приложение дальше, тестируя самые сложные функции с более высокими (иногда максимальными) значениями, чтобы определить скорость обработки программного обеспечения.

Тур “после работы’’ – после денежной активности приложения, оно все еще продолжает работу. Это поддержка задач, архивация данных и файлов. Это может быть автоматическим процессом, но его также можно выполнять принудительно. Данный тур напоминает нам делать это.

Тур FedEx – думайте о данных, которые должны проходить через приложение. Данные начинают свой путь с ввода, далее хранятся во внутренних хранилищах и переменных, где часто происходит их обработка и изменение, а далее они могут подвергаться вычислениям. В конце большинство из таких данных предоставляются пользователю или куда-либо еще. Нужно концентрироваться на данных. Нужно определить сохраненные данные и «проследить» их путь через приложение. Вот ввели вы, к примеру, адрес своей почты. А где он отображается? В каких местах? Попробуйте найти все области, с которыми соприкасаются данные. Определите весь их жизненный цикл.

Тур уборщика – Мы можем наметить места для методичной проверки интерфейса: экран за экраном, диалог за диалогом (преимущественно, по короткому пути) без детального тестирования. Только очевидные места. Указанный тур включает выбор цели (все меню, ошибки, диалоговые окна, к примеру) и посещение каждой из них наиболее коротким путем.

1. **Исследовательское тестирование ПО (Туры, Туры историческому району, Туры по району развлечений)**

Исторический район - Такие районы представляют legacy код, функции, представленные в старых версиях и исправленные ошибки, которые все также нужно проверять.

Музейный тур – музейные древности здесь — legacy код. Его можно определить по датам, указанной в репозитории. Такой код, использованный в новом окружении, склонен к плохенькой работе. Исследовать этот код сложно: программист может быть уже не в компании, а документация может быть скудной. Тестировщик должен выявить такой код и обратить внимание на его работу.

Путешествие по нерекомендуемым местам – в этом туре тестировщик фокусируется на проверке и выявлении проблемных или нежелательных аспектов программного обеспечения. Он активно ищет ошибки, уязвимости и проблемы безопасности, чтобы обеспечить надежность и защиту программы в неблагоприятных условиях.

Путешествие по уже посещенным местам – в этом туре тестировщик возвращается к предыдущим версиям программного обеспечения, которые уже были протестированы, чтобы проверить, что ранее обнаруженные проблемы были исправлены и новые функции работают без сбоев.

Тур по району развлечений – Это возможность немного развлечься с приложением: сделать его более привлекательным, поиграть с шаблонами, цветами. Это дает некоторый перерыв в усердной работе.

Путешествие актера второго плана – в этом туре тестировщик фокусируется на проверке и поддержке второстепенных или менее видимых функций и компонентов программного обеспечения, чтобы убедиться, что они работают правильно и не влияют на основные функции.

Путешествие по темным переулкам – в данном туре тестировщик исследует и проверяет скрытые или менее известные уголки программного обеспечения, где могут скрываться проблемы или уязвимости. Он активно ищет и исправляет потенциальные проблемы безопасности или ошибки.

Путешествие любителя ночной жизни – в данном туре тестировщик проводит тестирование программного обеспечения в условиях, когда оно работает непрерывно и под нагрузкой в течение длительного времени. Он проверяет стабильность, производительность и отклик системы в условиях непрерывной работы.

1. **Исследовательское тестирование ПО (Туры, Туры по туристическому району, Туры по району отелей)**

Тур по туристическому району – тур концентрируется на быстрых тестах, основанных на функционале, с которым столкнется новый пользователь, который нуб в компуктере.

Путешествие одинокого бизнесмена – в данном туре тестировщик фокусируется на проверке и обеспечении функциональности программного обеспечения в условиях одиночества и ограниченной доступности для помощи. Он проверяет, насколько легко и эффективно программное обеспечение может быть использовано и управляться без помощи других людей.

Путешествие супермодели – в этом туре тестировщик фокусируется на проверке и оценке пользовательского интерфейса программного обеспечения. Он стремится к элегантности, привлекательности и удобству использования пользовательского интерфейса, чтобы создать приятный и привлекательный пользовательский опыт.

Путешествие по шотландским пабам – это особенно актуально для крупных приложений. В них есть такие места, которые достаточно сложно найти без чьей-либо помощи. Их не сложно использовать, их просто сложно отыскать. Таких мест может быть действительно много. Проблема в том, чтобы узнать о них.

Путешествие коллекционера – коллекционируйте выходные данные, и чем больше, тем лучше. Идея в том, чтобы пройти туда, куда вы только можете и задокументировать все то, что вы увидите. Убедитесь, что вы увидите все выходные данные, которые может сгенерировать приложение. Для текстового редактора: печать, проверка орфографии, форматирование, различные структуры, таблицы, графика. Для магазина: возможность покупки из любого допустимого отдела, удачные и неудачные транзакции по картам. Нужно преследовать все возможные выходные данные, пока вы не сможете утвердить, что вы были везде, видели все и завершили свою коллекцию.

Тур по району отелей – это место, которое находится далеко от шума и суеты, место для легкого отдыха и расслабления. Это место, где можно отойти от основных функций приложения, популярных фич, и протестировать второстепенные, поддерживающие функции, которые могут быть не отражены в тест планах.

Путешествие под дождем – идея в том, чтобы начать операцию и затем резко ее остановить, словно в этот момент пошел дождь и вам не хочется ходить куда-либо в это время. Начните вводить информацию о покупке с тем, чтобы сразу закончить эту процедуру и выйти; начните печать и отмените ее до того, как она распечатается. Если функционал имеет кнопку отмены или же если он выполняется дольше, чем несколько секунд – используйте этот тур.

Путешествие лежебоки – Идея тура в том, чтобы делать настолько мало усилий, насколько это возможно. Оставляйте поля пустыми, принимайте значения по умолчанию, заполняйте формы наименьших количеством данных, не нажимайте ни на что дополнительное, пропускайте формы без нажатий или подайте туда все, что угодно. Если есть возможность выбора, выбирайте путь наименьшего сопротивления. Опять же, ленивое взаимодействие пользователя не значит, что приложение работает лениво: оно должно подставлять значения по умолчанию, обрабатывать пустые формы и т.д.

1. **Исследовательское тестирование ПО (Туры, Туры по неблагополучному району)**

Тур по неблагополучному району – входные данные должны ломать приложение.

Путешествие саботажника – Пытайтесь поломать приложение любыми возможными путями. Попросите приложение прочитать какие-либо данные с диска, и далее саботируйте эту попытку, навредив операции и файлу. Попросите сделать какую-либо операцию, которая интенсивно использует память, когда приложение установлено на машине с малым количеством памяти, или, когда одновременно запущено другое приложение, потребляющее большое количество ресурсов.

Антисоциальное путешествие – в данном районе происходят попытки разрушить приложение. Быть милым, вежливым и следовать толпе – не лучший способ сделать это. Нужно делать все наоборот.

Путешествие с неправильным поворотом – идея в том, чтобы делать что-то в неправильном порядке. Возьмите набор корректных действий и совершите их в некорректном порядке. Попробуйте оформить заказ до того, как положить что-либо в корзину. Попробуйте вернуть заказ до того, как вы его купили. Измените настройки доставки до того, как закончена покупка.

Криминальное путешествие – тур преступления. Идея в том, чтобы предоставить такие данные, которые вообще не должны появляться… ну вообще не должны. Вы не ожидаете, что турист украдет пинту пива в пабе? Это не просто антисоциальные действия, это преступления. Нарушая закон, турист попадет в неприятности. Нарушая закон, тест-инженер может встретить сообщения об ошибке. Данные в этом туре вызывают сообщения об ошибках, а если их нет – вы нашли баг. Вводите неправильные данные, неправильного формата, слишком длинные, слишком короткие и т.д.

Путешествие человека, склонного к повторяющимся действиям – в этом туре нужно вводить одни и те же данные снова и снова, выполнять одно и то же действие опять и опять. Копируйте, вставляйте, вырезайте, заменяйте, а потом сделайте это еще раз. Название этому, обычно, — репетиция. Выберите товар, а потом снова этот же товар, чтобы проверить появляются ли скидки.

1. **Тестирование требований (Почему важно? Что тестируется? Параметры тестирования документации)**

Требование – это описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи. До 30% ошибок появляются из-за некорректных требований.

**Важность требований:**

* Позволяют понять, что и с соблюдением каких условий система должна делать.
* Предоставляют возможность оценить масштаб изменений и управлять изменениями.
* Являются основой для формирования плана проекта (в том числе плана тестирования).
* Помогают предотвращать или разрешать конфликтные ситуации.
* Упрощают расстановку приоритетов в наборе задач. Позволяют объективно оценить степень прогресса в разработке проекта.

**Тестируется:**

* Продуктная документация (план проекта, спецификации, архитектура, дизайн и т.д.)
* Проектная документация (сопроводительная, маркетинговая и т.д.)

**Параметры тестирования документации:**

* Четкость и ясность
* Актуальность
* Логика
* Возможные сценарии
* Интеграция

**Основные принципы:**

* Проводится до старта разработки
* Проводится как аналитиками, так и тестировщиками
* Баги заводятся так же как и любые другие
* Оповещение разработчиков (если проводится на этапе разработки)

1. **Тестирование требований (Уровни и виды требований, Критерии качества требований, Методы тестирования требований)**

**Уровни и виды требований:**

* Бизнес-требования – выражают цель, ради которой разрабатывается продукт (зачем он нужен, какая от него ожидается польза). Результатом выявления требований на этом уровне является общее видение (vision and scope) — документ, который, как правило, представлен простым текстом и таблицами. Нет детализации поведения системы и иных технических характеристик, но могут быть определены приоритеты решаемых бизнес-задач, риски и т.п.
* Пользовательские требования – описывают задачи, которые пользователь может выполнять с помощью разрабатываемой системы (реакцию системы на действия пользователя, сценарии работы пользователя). Т.к. появляется описание поведения системы, требования этого уровня могут быть использованы для оценки объёма работ, стоимости проекта, времени разработки и т.д.
* Бизнес-правила – описывают особенности принятых в предметной области (и/или непосредственно у заказчика) процессов, ограничений и иных правил. Эти правила могут относиться к бизнес-процессам, правилам работы сотрудников, нюансам работы ПО и т.д.
* Функциональные требования – описывают поведение системы, т.е. её действия (вычисления, преобразования, проверки, обработку и т.д.). В контексте проектирования функциональные требования в основном влияют на дизайн системы.
* Нефункциональные требования – описывают свойства системы (удобство использования, безопасность, надёжность, расширяемость и т.д.), которыми она должна обладать при реализации своего поведения. Здесь приводится более техническое и детальное описание атрибутов качества. В контексте проектирования нефункциональные требования в основном влияют на архитектуру системы.
* Требования к интерфейсам – описывают особенности взаимодействия разрабатываемой системы с другими системами и операционной средой.
* Требования к данным – описывают структуры данных (и сами данные), являющиеся неотъемлемой частью разрабатываемой системы. Часто сюда относят описание базы данных и особенностей её использования.

**Критерии качества требований:**

* Корректность
* Недвусмысленность
* Полнота
* Непротиворечивость
* Упорядоченность по важности и стабильности
* Проверяемость
* Модифицируемость
* Трассируемость

**Методы тестирования требований:**

* Метод просмотра
* Метод экспертизы
* Метод составления вариантов тестирования

1. **Нефункциональное тестирование (Что это, Его параметры, и виды. Инсталляционное тестирование)**

Нефункциональное тестирование – тестирование, которое проводится для оценки характеристик программного обеспечения. Проверяется не корректность работы функций приложения, а сопутствующие характеристики.

**Параметры:**

* Безопасность
* Надежность
* Живучесть
* Наличие (зависимость от системы)
* Удобство использование
* Масштабируемость
* Совместимость
* Эффективность
* Гибкость
* Портативность
* Возможность повторного использования

**Виды:**

* Инсталяционное тестирование
* Тестирование производительности
* Тестирование безопасности
* Тестирование эргономичности
* Тестирование совместимости
* UI тестирование
* Тестирование локализации и интернационализации
* A/B тестирование
* Тестирование на отказ и восстановление
* Тестирование на соответствие стандартам
* Тестирование на прерывания (Мобильное ПО)
* Тестирование соединения (Мобильное ПО*)*

Инсталяционное тестирование – тестирование уровня корректности установки некоего программного продукта в искусственно созданной среде с целью выявления степени ее готовности к эксплуатации.

Что необходимо учитывать при создании инсталлятора:

* Глубокое взаимодействие с ОС
* Совместимость библиотек, компонент или драйверов с разными платформами
* Удобство использования
* Дизайн и стиль
* Совместимость настроек и документов в разных версиях приложения.

Риски:

* Риск потери пользовательских данных
* Риск вывода ОС из строя
* Риск неработоспособности приложения
* Риск некорректной работы приложения

1. **Нефункциональное тестирование (Тестирование производительности)**

Тестирование производительности – комплекс тестов определяющих поведение приложения при нормальной и предельной нагрузках.

**Подвиды:**

* Тестирование производительности – тестирование, которое проводится с целью определения, как быстро работает вычислительная система или её часть под определённой нагрузкой (определение количества пользователей, могущих одновременно работать с приложением, измерение времени выполнения операций).
* Нагрузочное тестирование – тестирование, которое проводится для того, чтобы оценить поведение приложения под заданной ожидаемой нагрузкой. Этой нагрузкой может быть, например, ожидаемое количество одновременно работающих пользователей приложения, совершающих заданное число транзакций за интервал времени.
* Стресс-тестирование – тестирования проводится для определения надёжности системы во время экстремальных или диспропорциональных нагрузок и отвечает на вопросы о достаточной производительности системы в случае, если текущая нагрузка сильно превысит ожидаемый максимум.
* Тестирование стабильности – тестирование стабильности проводится с целью убедиться в том, что приложение выдерживает ожидаемую нагрузку в течение длительного времени. Кроме того, такое тестирование выявляет деградацию производительности, выражающуюся в снижении скорости обработки информации и/или увеличении времени ответа приложения после продолжительной работы по сравнению с началом теста.
* Конфигурационное тестирование – вариант тестирования программного обеспечения на компьютере с различными настройками аппаратного или программного обеспечения, а именно операционной системы, браузера, поддерживаемых драйверов и т.д.

1. **Нефункциональное тестирование (Тестирование эргономичности, Тестирование совместимости, Тестирование GUI)**

Тестирование эргономичности – исследование, выполняемое с целью определения, удобен ли некоторый искусственный объект (такой как веб-страница, пользовательский интерфейс или устройство) для его предполагаемого применения. Это метод оценки удобства продукта в использовании, основанный на привлечении пользователей в качестве тестировщиков, испытателей и суммировании полученных от них выводов.

Тестирование совместимости – тестирование, целью которого является проверка корректной работы приложения в определенном окружении (аппаратная платформа, сетевые устройства, ОС, периферия, БД, браузеры).

Тестирование GUI – тестирование, проверяющее соответствие внешнего вида продукта заявленным дизайнам и требованиям.

1. **Нефункциональное тестирование (Тестирование глобализации, A/B тестирование, Тестирование на отказ и восстановление системы, Тестирование на соответствие стандартам)**

**Тестирование глобализации** – тестирование проверяющее адаптацию приложения к разным регионам и странам.

**Локализация программного обеспечения**

Процесс адаптации программного обеспечения к культуре какой-либо страны. Как частность — перевод пользовательского интерфейса, документации и сопутствующих файлов программного обеспечения с одного языка на другой.

**Интернационализация программного обеспечения**

Технологические приёмы разработки, упрощающие адаптацию продукта к языковым и культурным особенностям региона (регионов), отличного от того, в котором разрабатывался продукт.

**Отличие локализации от интернационализации:**

Интернационализация производится на начальных этапах разработки, в то время как локализация — для каждого целевого языка

**A/B тестирование** – метод маркетингового исследования, суть которого заключается в том, что контрольная группа элементов сравнивается с набором тестовых групп, в которых один или несколько показателей были изменены, для того, чтобы выяснить, какие из изменений улучшают целевой показатель.

**Тестирование на отказ и восстановление системы** – тестирование, которое проверяет продукт с точки зрения способности противостоять и успешно восстанавливаться после возможных сбоев, возникших в связи с ошибками программного обеспечения, отказами оборудования или проблемами связи. Целью данного вида тестирования является проверка систем восстановления, которые, в случае возникновения сбоев, обеспечат сохранность и целостность данных тестируемого продукта.

**Тестирование на соответствие стандартам** – Процесс тестирования для определения соответствия компонента или системы стандартам, нормам и правилам (Apple guidelines, Google Play guidelines, ISO/IEC 25010:2011).

1. **Тестирование безопасности (Что это, Причины пробелов в безопасности, Тестирование проницаемости)**

Тестирование безопасности – вид нефункционального тестирования, направленный на оценку уязвимости программного обеспечения к различным атакам.

Уязвимость – это недостаток или слабость в проектировании, реализации, эксплуатации или управлении системы, которые могут быть использованы для компрометации целей безопасности системы.

Угроза – это что угодно (вредоносный внешний злоумышленник, внутренний пользователь, нестабильность системы и т. д.), что может нанести ущерб частям приложения (ценным ресурсам, таким как данные в базе данных или в файлах системы), используя уязвимость.

**Причины появления пробелов:**

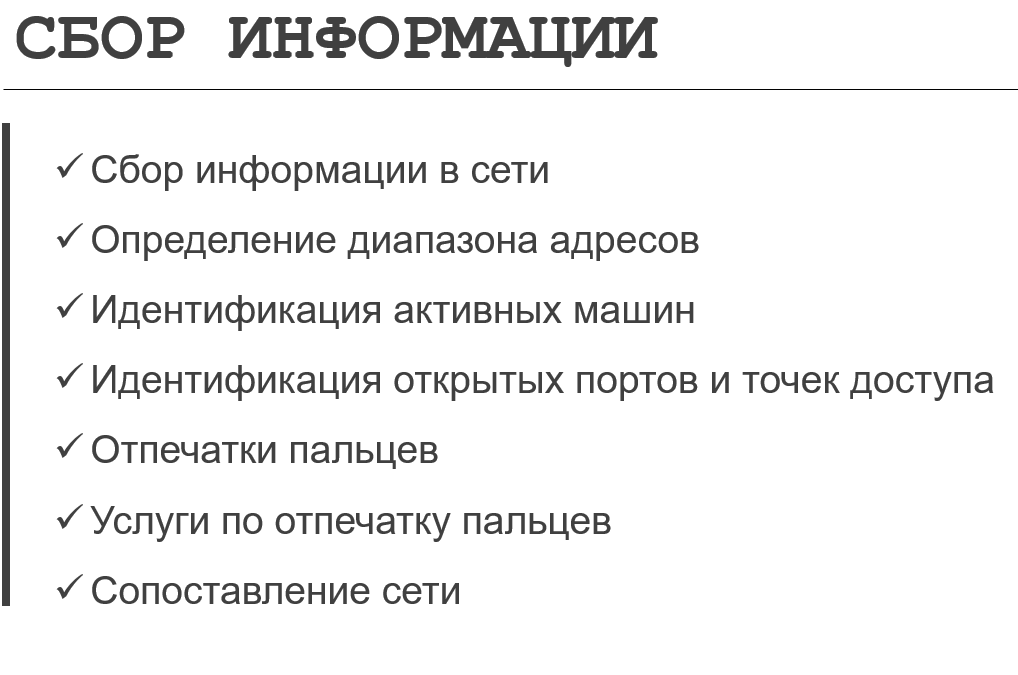
* Ошибки проектирования
* Некорректная настройка взаимодействия оборудования и ПО
* Проблемы подключения
* Человеческий фактор
* Погрешность коммуникации
* Чрезмерная сложность системы
* Недостаточная квалификация сотрудников

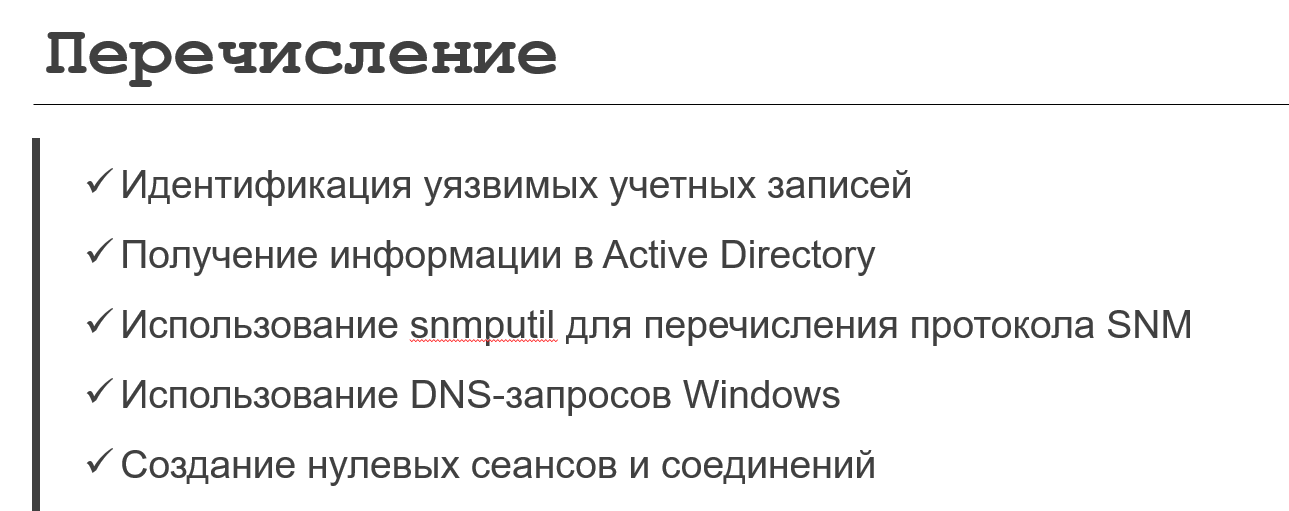
**Простейшие способы взлома:**

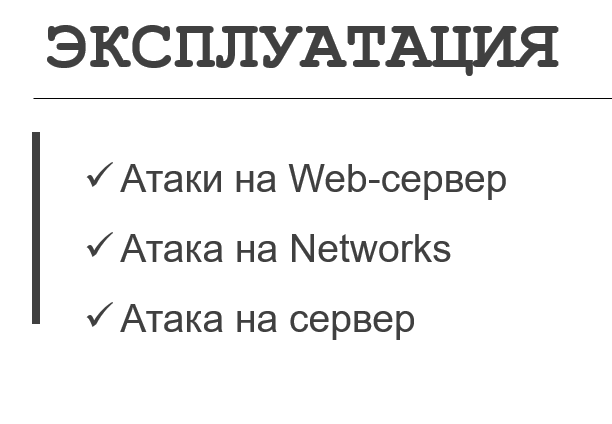
* Попытка узнать пароль с помощью внешних средств
* Атака системы с помощью утилит
* Подавление, ошеломление системы
* Целенаправленное введение ошибок
* Просмотр несекретных данных

**Тесты на проникновение:**

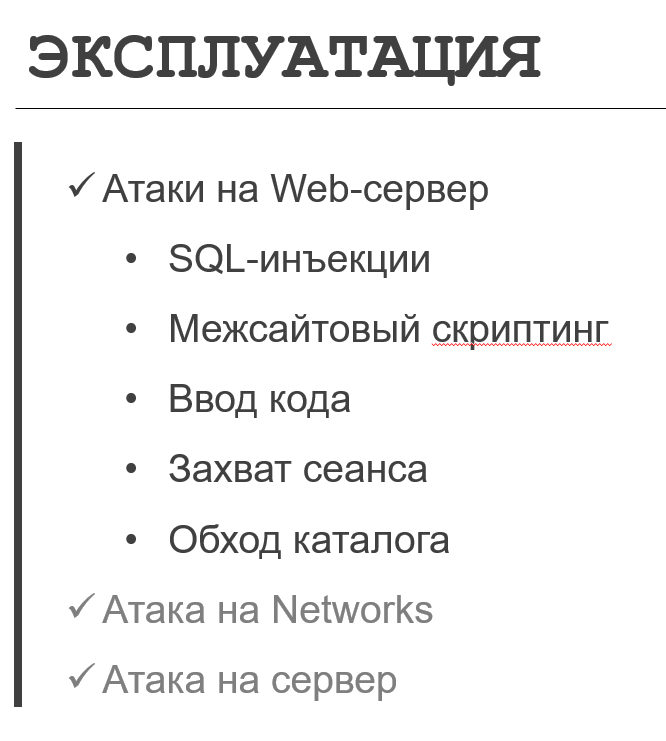
* Сбор информации
* Сканирование
* Перечисление
* Эксплуатация



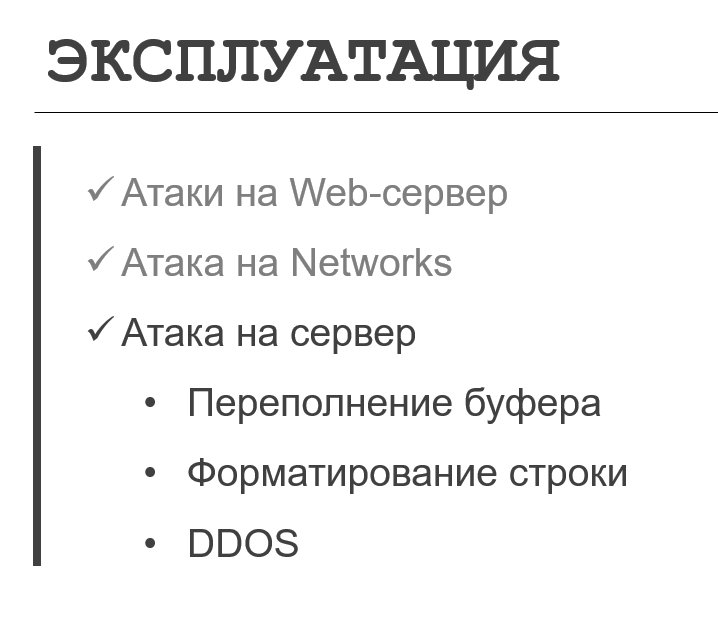




1. **Тестирование безопасности (Виды атак)**







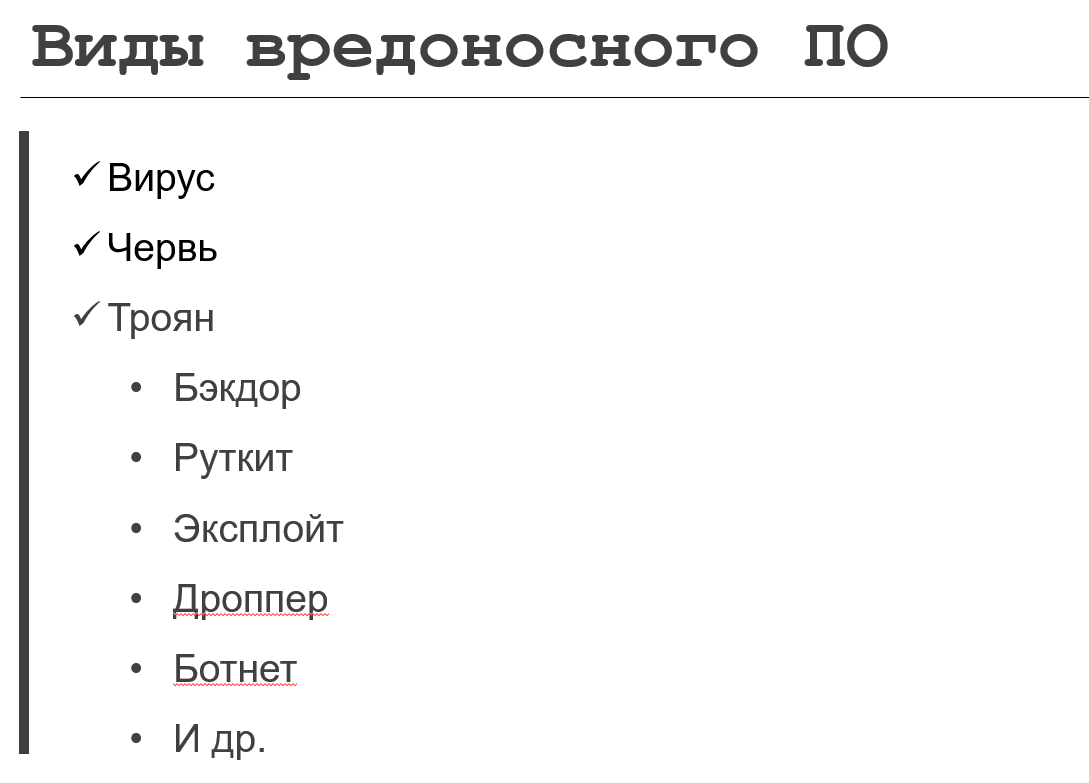
SQL-инъекции: Атака, при которой злоумышленник внедряет вредоносные SQL-запросы с целью получения несанкционированного доступа к базе данных.

Командные инъекции: Злоумышленник внедряет команды в системные вызовы или командные интерпретаторы для выполнения вредоносных действий.

Атаки, направленные на использование слабых или неправильных конфигураций системы, такие как открытые порты, неправильные права доступа и другие настройки.

DDoS-атаки на отказ в обслуживании: Направлены на перегрузку ресурсов системы, чтобы она стала недоступной для легитимных пользователей.

1. **Тестирование безопасности (Виды вредоносного ПО)**



Вирус - это вредоносная программа, которая способна копировать себя и внедряться в другие исполняемые файлы или системные области памяти. Когда зараженный файл или программа выполняется, вирус может выполнять различные вредоносные действия, такие как уничтожение данных, блокировка системы или распространение на другие компьютеры через сеть.

Червь - это вирус, который способен самостоятельно распространяться по компьютерным сетям, внедряясь в уязвимые узлы без необходимости взаимодействия пользователя. Черви могут использовать различные методы распространения, такие как использование слабых паролей, эксплойтов уязвимостей или подделку электронных писем.

Троян - это вредоносная программа, которая маскируется под полезное или легитимное приложение, чтобы обмануть пользователя и получить несанкционированный доступ к системе или данные пользователя. Трояны могут выполнять различные действия, включая сбор личной информации, удаленное управление системой, установку дополнительных вредоносных программ и т. д.

Бэкдор - это скрытый механизм или программа, предназначенные для обхода нормальных механизмов аутентификации и обеспечения несанкционированного доступа к системе. Бэкдоры могут быть использованы злоумышленниками для удаленного управления компьютером или для выполнения вредоносных действий без ведома пользователя.

Руткит - это набор вредоносных программ и техник, предназначенных для скрытия наличия вредоносного программного обеспечения на зараженной системе. Руткиты могут изменять ядро операционной системы или другие системные компоненты, чтобы скрыть свою активность от обычных механизмов обнаружения и удаления вредоносных программ.

Эксплоит - это программа или код, который использует известную уязвимость в программном обеспечении или системе для выполнения вредоносных действий или получения несанкционированного доступа к системе. Эксплоиты могут быть использованы злоумышленниками для атаки на уязвимые системы и приложения.

1. **Тестирование безопасности (Тестирование на проникновение)**

естирование на проникновение (penetration testing), также известное как тестирование проницаемости или этическое взлом, является процессом активного анализа системы, приложения или сети с целью выявления уязвимостей и оценки их потенциального воздействия на систему. Главной целью тестирования на проникновение является проверка безопасности системы и идентификация слабых мест, которые могут быть использованы злоумышленниками для несанкционированного доступа или атак.

Вот некоторые методы и шаги, которые могут быть использованы при проведении тестирования на проникновение:

1. Сбор информации (Reconnaissance): Этот шаг включает сбор информации о целевой системе или сети, такой как IP-адреса, доменные имена, информацию о сотрудниках и т. д. Это может включать использование открытых источников, сканирование портов, поиск уязвимостей и других методов сбора информации.
2. Анализ уязвимостей (Vulnerability Analysis): В этом шаге тестировщик исследует систему или сеть, используя различные инструменты и методы, чтобы выявить уязвимости. Это может включать сканирование уязвимостей, анализ кода, проверку конфигурации, анализ безопасности приложений и другие техники.
3. Эксплуатация уязвимостей (Exploitation): Если уязвимости были обнаружены, тестировщик пытается эксплуатировать их, чтобы получить несанкционированный доступ или выполнить другие вредоносные действия. Например, это может включать попытки взлома паролей, использование известных уязвимостей, выполнение кода на удаленной системе и т. д.
4. Поддержание доступа (Maintaining Access): Если тестировщик получает несанкционированный доступ к системе, он может попытаться поддерживать этот доступ для дальнейшего изучения системы и выполнения дополнительных атак.
5. Получение привилегий (Privilege Escalation): В этом шаге тестировщик пытается получить повышенные привилегии или административный доступ к системе, чтобы иметь больше возможностей для атаки.
6. Составление отчета (Reporting): По завершении тестирования на проникновение составляется детальный отчет, в котором описываются обнаруженные уязвимости, успешность эксплуатации, потенциальные последствия и рекомендации по устранению проблем.
7. **Тестовая документация (Ее виды и назначение, Тест план)**

Тестовая документация – это документация, создаваемая тестировщиками, которая помогает в выполнении различного рода активностей в рамках тестирования программного обеспечения.

**Виды:**

* Тест план
* График тестирования
* Матрица устройств
* Матрица прослеживаемости
* Тестовый набор
* Тест сценарии
* Тест кейсы
* Чеклист

**Отчетность:**

* Отчеты об ошибках
* Отчеты о результатах тестирования

Тест план – документ, описывающий весь объем работ по тестированию, начиная с описания тестируемых объектов, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

**Цель написания тест плана:**

* Продумать стратегию тестирования программного обеспечения;
* Описать процесс тестирования на проекте, и как он встраивается в процесс разработки;
* Обеспечить информированность каждого члена команды об активностях QA команды, распределении обязанностей и зон ответственности;
* Скорректировать ожидания заказчика от команды тестирования.

1. **Тестовая документация (График тестирования, Матрица трассируемости, Тестовый набор, Чек-лист)**

**График тестирования** – документ описывающий последовательность выполнения активностей по тестированию членами QA команды, с указанием дат начала выполнения работ и их завершения.

Цель создания графика:

* Согласовать работу команды разработки и тестирования;
* Обеспечить информированность каждого члена команды о последовательности задач, а также о сроках их выполнения;
* Обеспечить прозрачность процесса тестирования для заказчика;
* Обеспечить возможность отслеживания отставаний от плана и влияния добавления дополнительных задач команде.

**Матрица устройств** – документ, используемый на проектах, где разрабатываются мобильные приложения, который описывает конфигурацию устройств, где будет проводиться тестирование.

Цель создания матрицы устройств:

* Определить оптимальное количество устройств, необходимых для тестирования;
* Убедится, что устройства с различными версиями необходимого оборудования участвуют в тестировании;
* Сосредоточить усилия тестировщика на устройствах, оптимально полно покрывающих требования проекта;
* Обеспечить прозрачность тестирования для заказчика.

**Матрица прослеживаемости** – документ, используемый для определения покрытия требований тестовой документацией.

Цель создания матрицы прослеживаемости:

* Обеспечить должное покрытие всех функциональных и нефункциональных требований тестами.

**Тестовый набор** – документ, вмещающий в себя набор тестов/тестовых случаев/тестовых сценариев.

Цель создания тестового набора:

* Сгруппировать тестовые случаи по:
  + Видам тестирования;
  + Уровням тестирования;
  + Приоритету;
* Облегчить распределение объема тестирования в команде;
* Облегчить оценку трудозатрат тестировщика.

**Чеклист** – документ, перечисляющий идеи для проверки. Документ, который очень поверхностно указывает, что необходимо проверить в приложении, но не указывает, как это сделать.

Виды тестирования для которых пишут чеклисты:

* Тестирование совместимости;
* Тестирование инсталляции продукта;
* Исследовательское тестирование;
* CRUD тестирование
* Тестирование на прерывания и т.д.

1. **Техники тест дизайна (Что это, Методы черного ящика: классы эквивалентности, Анализ граничных значений)**

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

Метод черного ящика – Техника базируется на анализе спецификаций или других проектных документов, описывающих требования к системе.

**Разработка тестов методом черного ящика:**

* Для описания задач, которые должны быть решены, программных продуктов или их компонентов, используются модели - формальные или неформальные;
* Из этих моделей систематически выводятся тестовые сценарии.

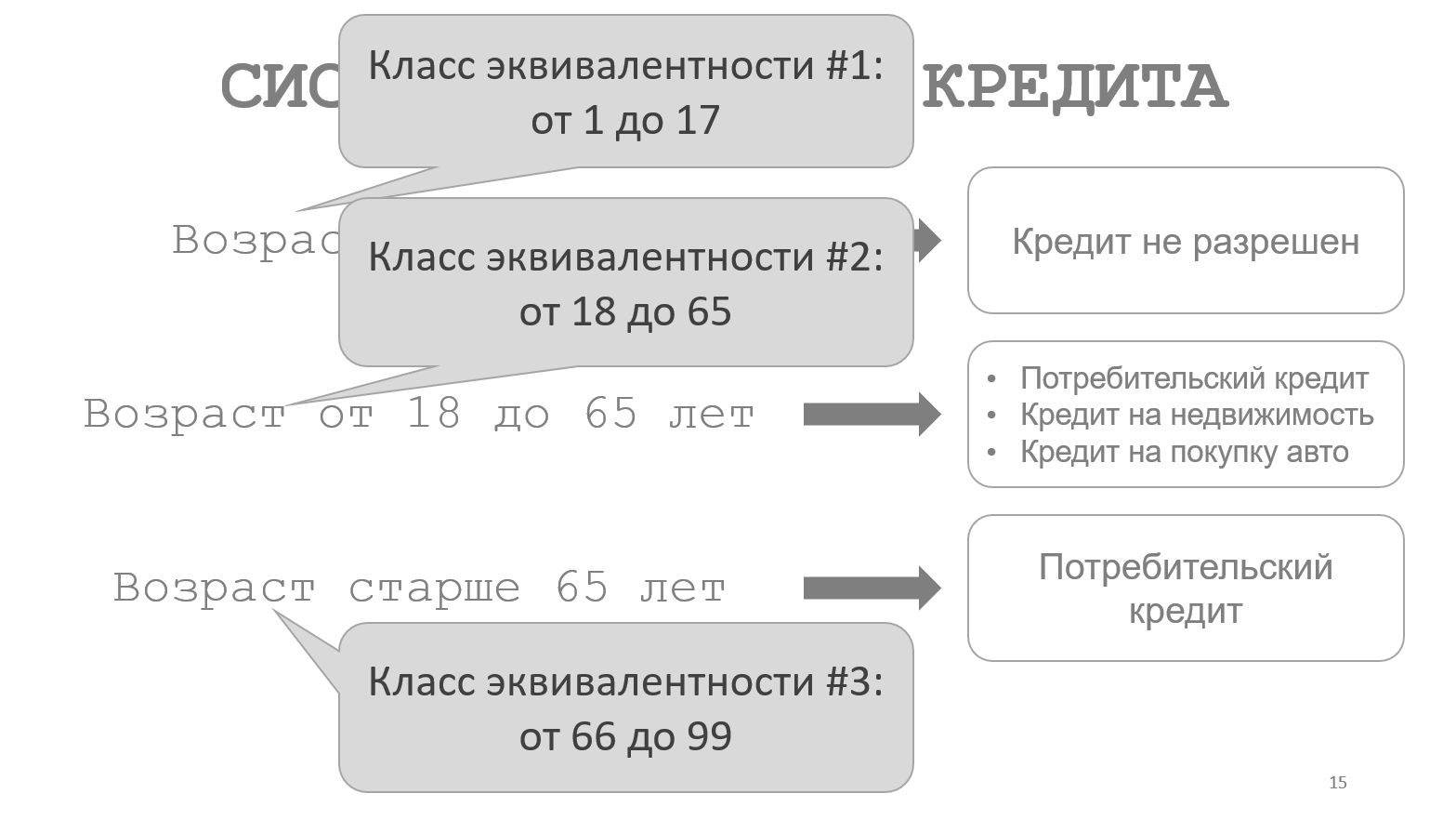
**ТД методом черного ящика:**

* Эквивалентное разбиение;
* Анализ граничных значений;
* Тестирование таблицы решений;
* Тестирование таблицы переходов;
* Тестирование по сценариям использования.

**ЭКВИВАЛЕНТНОЕ РАЗБИЕНИЕ**

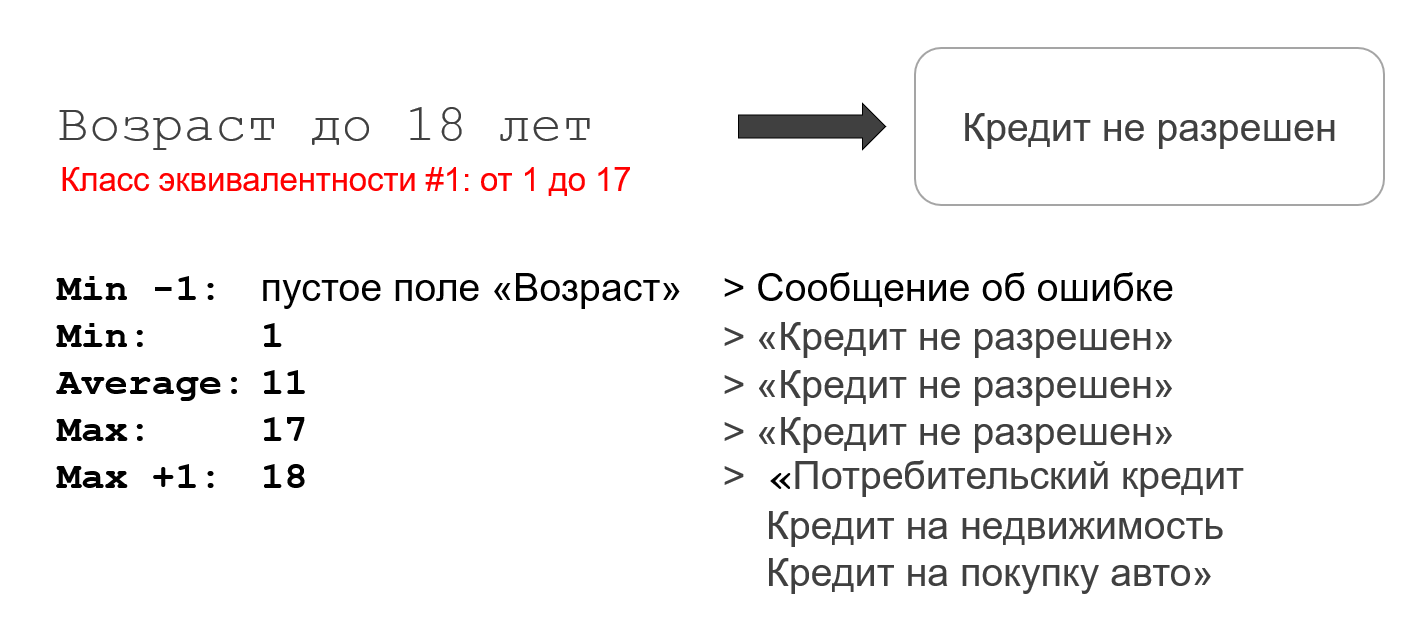
Входные данные для программного обеспечения или системы разбиваются на **группы**, от которых ожидается сходное поведение, то есть они должны обрабатываться аналогичным образом.

Данные группы называются **классами эквивалентности**.



**АНАЛИЗ ГРАНИЧНЫХ ЗНАЧЕНИЙ**

Поведение на границах эквивалентных областей имеет наибольшие шансы быть некорректным, таким образом границы являются потенциальным источником дефектов. Минимальные и максимальные значения сегмента являются граничными значениями.



1. **Техники тест дизайна (Методы черного ящика: таблица решений, таблица (диаграмма) переходов)**

**ТЕСТИРОВАНИЕ ТАБЛИЦЫ РЕШЕНИЙ**

Таблица решений содержит триггерные условия, обычно комбинации значений «истина» и «ложь» для всех входных условий, и результирующие действия для каждой комбинации условий. Каждый столбец таблицы соотносится с бизнес-правилом, определяющим уникальную комбинацию условий и результат выполнения действий, связанных с этим правилом.

Стандартом покрытия для тестирования таблицы решений обычно является наличие хотя бы одного теста для каждой колонки, что обычно включает в себя покрытие всех комбинаций триггерных условий.

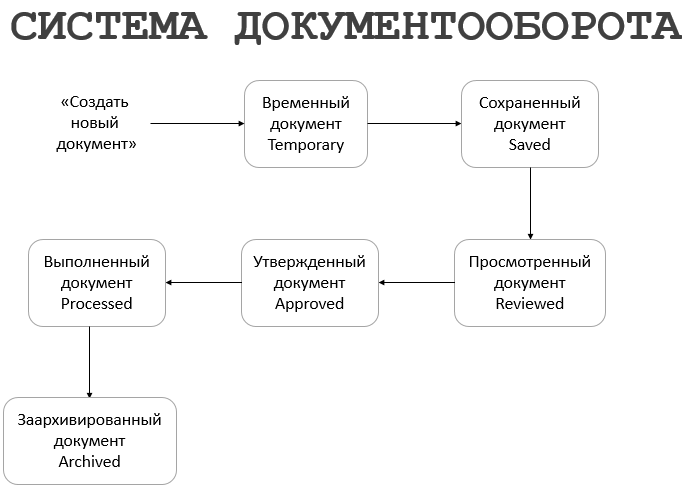




**ТЕСТИРОВАНИЕ ТАБЛИЦЫ ПЕРЕХОДОВ**

Система может показывать различные отклики в зависимости от текущих условий или предшествовавшей истории состояний. Данный метод позволяет тестировщику рассматривать систему с точки зрения её состояний, переходов между состояниями, входов или событий, активизирующих изменения состояний (переходы) и действия, к которым приводят эти переходы. Состояния системы или тестируемого объекта разделяемы, определяемы и конечны.

Таблица состояний демонстрирует связи между состояниями и входами и может подсказать возможные некорректные переходы.





1. **Техники тест дизайна (Методы черного ящика: тестирование по вариантам использования, попарное тестирование)**

**СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ**

Сценарий использования (use case) описывает взаимодействия между участниками (включая пользователей и систему) приводящие к полезным результатам для заказчика или пользователя системы. Сценарии использования могут быть описаны на уровне абстракций (бизнес сценарий использования, уровень бизнес-процессов, не связанный с технологией) или на системном уровне (сценарий использования системы на уровне системного функционала).

**ХОРОШИЙ ТЕСТ**

* Предположительно найдет ошибки
* Не избыточен
* Не слишком простой, не и не слишком сложный

В процессе создания прорабатываются основной, альтернативный и исключительный пути.

Критерий качества теста- комбинация критерия частоты и критерия важности.

**Действия при разработке сценария**

* Идентификация всех значений
* Выделение классов эквивалентности
* Построение таблиц с комбинациями значений из классов эквивалентности
* Написание тест-кейсов

**Ошибки при написании сценариев**

* Сценарий начинает система
* Система предсказывает действия пользователя
* Наличие внутренних сбоев
* Действия по устранению вместо альтернатив

**ПОПАРНОЕ ТЕСТИРОВАНИЕ**

Попарное тестирование – техника, которая предполагает, что тестовые сценарии разрабатываются таким образом, чтобы выполнить все возможные отдельные комбинации каждой пары входных параметров

Рассмотрим пример. Возьмем наушники с такими характеристиками:

* Тип подключения (беспроводной или проводной)
* Микрофон (включен или выключен)
* Подсветка (включена или выключена)



1. **Техники тест дизайна (Методы белого ящика: тестирование покрытия операторов, тестирование покрытия ветвей. Методы, основанные на опыте)**

Метод белого ящика – техника базируется на анализе внутренней структуры компонентов системы.

Разработка тестов методом белого ящика:

* Тестовые сценарии выводятся на основе информации о том, как спроектировано программное обеспечение (например, на основе программного кода и подробного описания проектного решения).
* Для программного обеспечения может быть измерена величина покрытия для имеющихся тестовых сценариев, и последующие тестовые сценарии могут разрабатываться для систематического увеличения покрытия.

ТД методом белого ящика:

* Компонентный уровень:

структура компонента программного обеспечения, т.е. операторы, альтернативы, ветви или определенные пути;

* Интеграционный уровень:

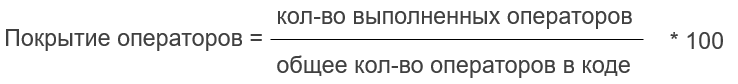
структура может быть представлена деревом вызовов (диаграмма, в которой модули вызывают другие модули);

* Системный уровень:

структура может представлять собой структуру меню, бизнес-процессов или же схему веб-страницы.

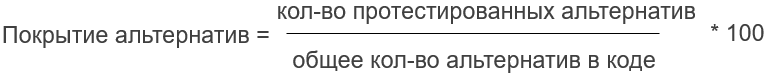
**ТЕСТИРОВАНИЕ ПОКРЫТИЯ ОПЕРАТОРОВ**

Техника тестирования, которая включает в себя выполнение всех операторов хотя бы единожды. Полученная метрика позволяет высчитать количество операторов, которые были выполнены и которые вообще есть в коде.



**ТЕСТИРОВАНИЕ ПОКРЫТИЯ АЛЬТЕРНАТИВ**

Техника тестирования, которая связана с тестированием ветвей (branches) (например, вариантов «True» и «False» для оператора «IF»). В методе тестирования альтернатив тестовые сценарии создаются для выполнения определенных результатов альтернатив. Ветви исходят из точек альтернатив в программном коде и показывают передачу управления различным участкам кода.



Тестирование альтернатив - это вид тестирования потока управления, так как оно описывает прохождение определенного потока через точки альтернативы. Покрытие альтернатив более строгое, чем покрытие операторов: 100% покрытие альтернатив обеспечивает 100% покрытие операторов, но не наоборот.

**ТЕСТИРОВАНИЕ ПОКРЫТИЯ УСЛОВИЙ**

Покрытие условий (condition) и покрытие множественных условий (multiple condition) - это более высокий уровень покрытия структуры кода после покрытия альтернатив. Техника предполагает покрытие всех условий, которые могут повлиять на результат принятия решения.

Метод создания тестов на основе опыта:

* Для определения тестовых сценариев используются человеческие знания и опыт.
* Знания тестировщиков, разработчиков, пользователей и заинтересованных лиц о программном продукте, его использовании и окружении, являются одним источником информации.
* Знания о вероятных дефектах и их распределении являются другим источником информации.

**ТЕСТ ДИЗАЙН НА ОСНОВЕ ОПЫТА**

* Предположения об ошибках (error guessing technique);
* Исследовательское тестирование (exploratory testing).

1. **Тестовые случаи (Цели, Жизненный цикл, Атрибуты тестовых случаев)**

Тестовый случай – формально описанный алгоритм тестирования программы, специально созданный для определения возникновения в программе определённой ситуации, определённых выходных данных.

**Цели:**

* Структурировать и систематизировать подход к тестированию
* Вычислять метрики тестового покрытия и принимать меры по его увеличению
* Отслеживать соответствие текущей ситуации плану
* Наладить взаимопонимание между заказчиком, разработчиками и тестировщиками
* Хранить информацию для длительного использования и обмена опытом между сотрудниками и командами
* Повышать качество требований
* Быстро вводить в курс дела нового сотрудника, недавно подключившегося к проекту.

**Жизненный цикл:**

* Создан (new)
* Запланирован (planned, ready for testing)
* Не выполнен (not tested)
* Выполняется (work in progress)
* Пропущен (skipped)
* Провален (failed)
* Пройден успешно (passed)
* Заблокирован (blocked)
* Закрыт (closed)
* Требует доработки (not ready)

**Атрибуты:**

* Идентификатор (ID);
* Связанное требование (requirement)
* Краткое название тестового случая (Summary, Title)
* Цель тестового случая (Goal, Aim, Description)
* Предусловия (Precondition)
* Шаги (Steps);
* Ожидаемый результат (Expected result);
* Постусловия (Postcondition);
* Статус (Status);
* Уровень (Level);
* Приоритет (Priority);
* Автор (Author)
* Комментарии (Notes, Comments)

1. **Тестовые случаи (Свойства качественных тестовых случаев, типичные ошибки при создании тестовых случаев)**

**Свойства:**

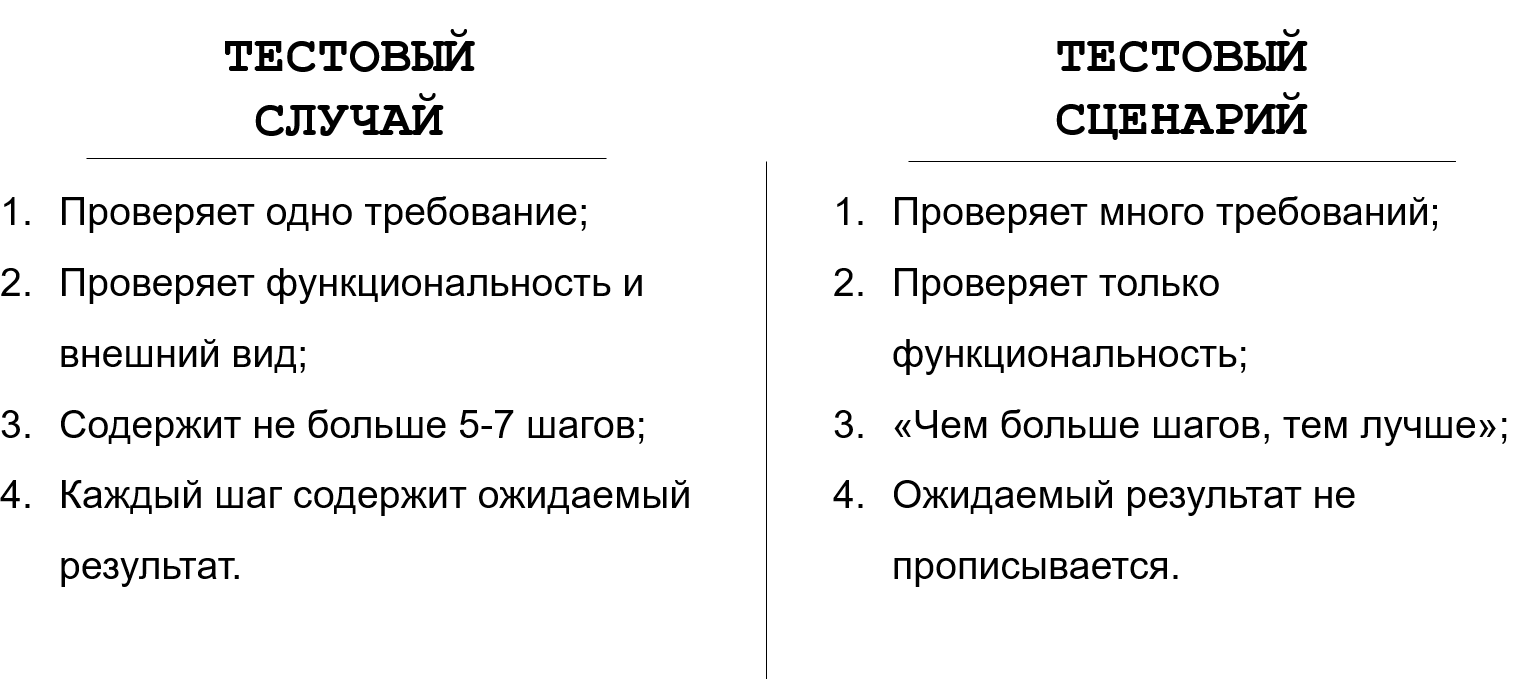
* Правильный технический язык и единообразие
* Баланс между специфичностью о общностью
* Баланс между простотой и сложностью
* Показательность
* Последовательность
* Отсутствие лишних действий
* Неизбыточность к другим тест-кейсам
* Демонстративность
* Прослеживаемость
* Повторяемость

**Типичные ошибки:**

* Отсутствие или плохо написанный Title
* Отсутствие нумерации шагов и ожидаемых результатов
* Ссылки на другие тест-кейсы
* Несоответствующая детализация
* Расплывчатые формулировки
* Описание событий или результатов в качестве шагов
* Выдумывание особенностей поведения приложения и предусловий
* Полное копирование других тест-кейсов на разных уровнях
* Непонимание принципов работы приложения
* Проверка типичной функциональности
* Ошибка как ожидаемый результат
* Общая некорректность

1. **Тестовые сценарии (Цели, Шаги для создания, Атрибуты тестовых сценариев)**

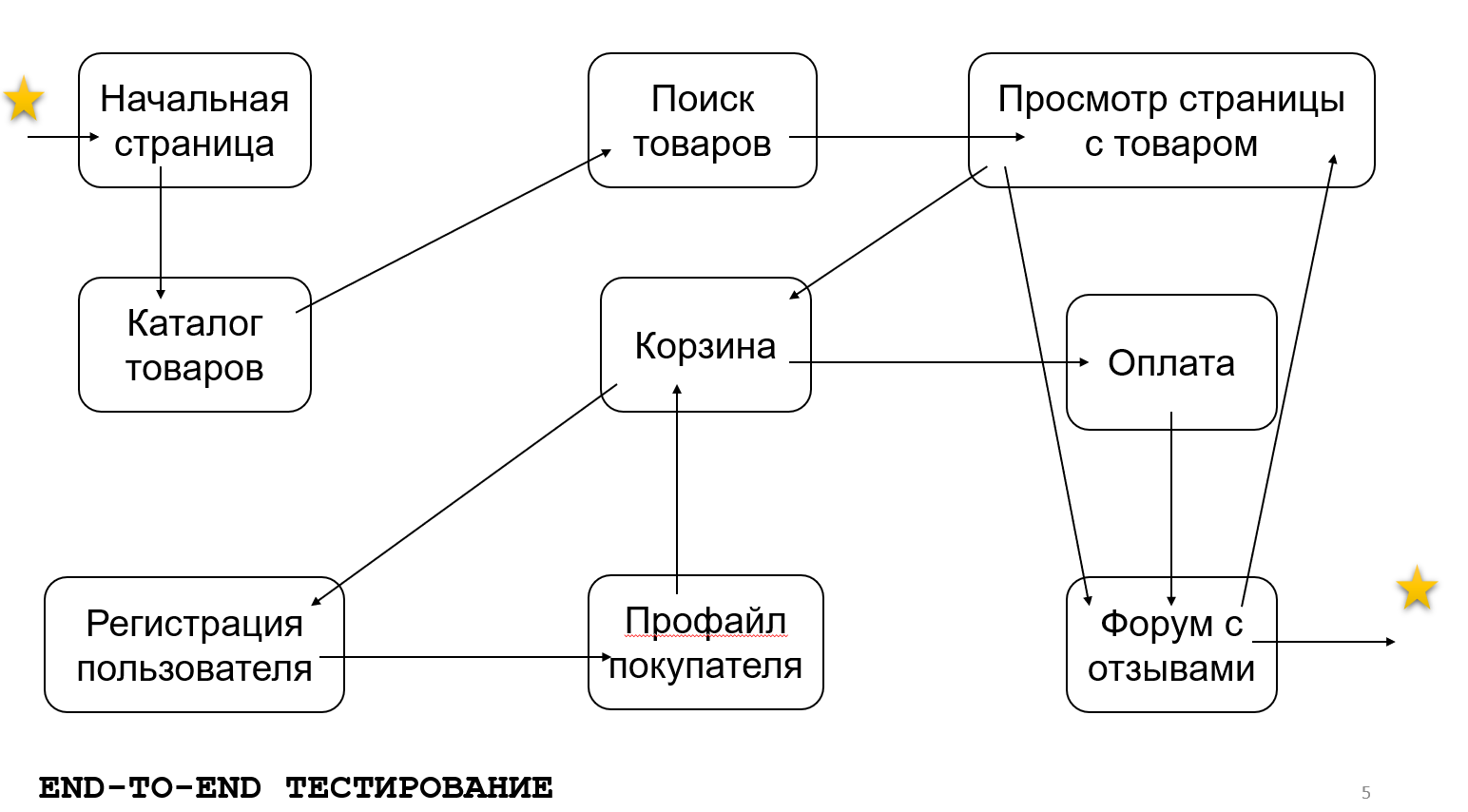
Тестовый сценарий – формально описанный алгоритм тестирования программы, специально созданный для определения возникновения в программе определённой ситуации, определённых выходных данных.

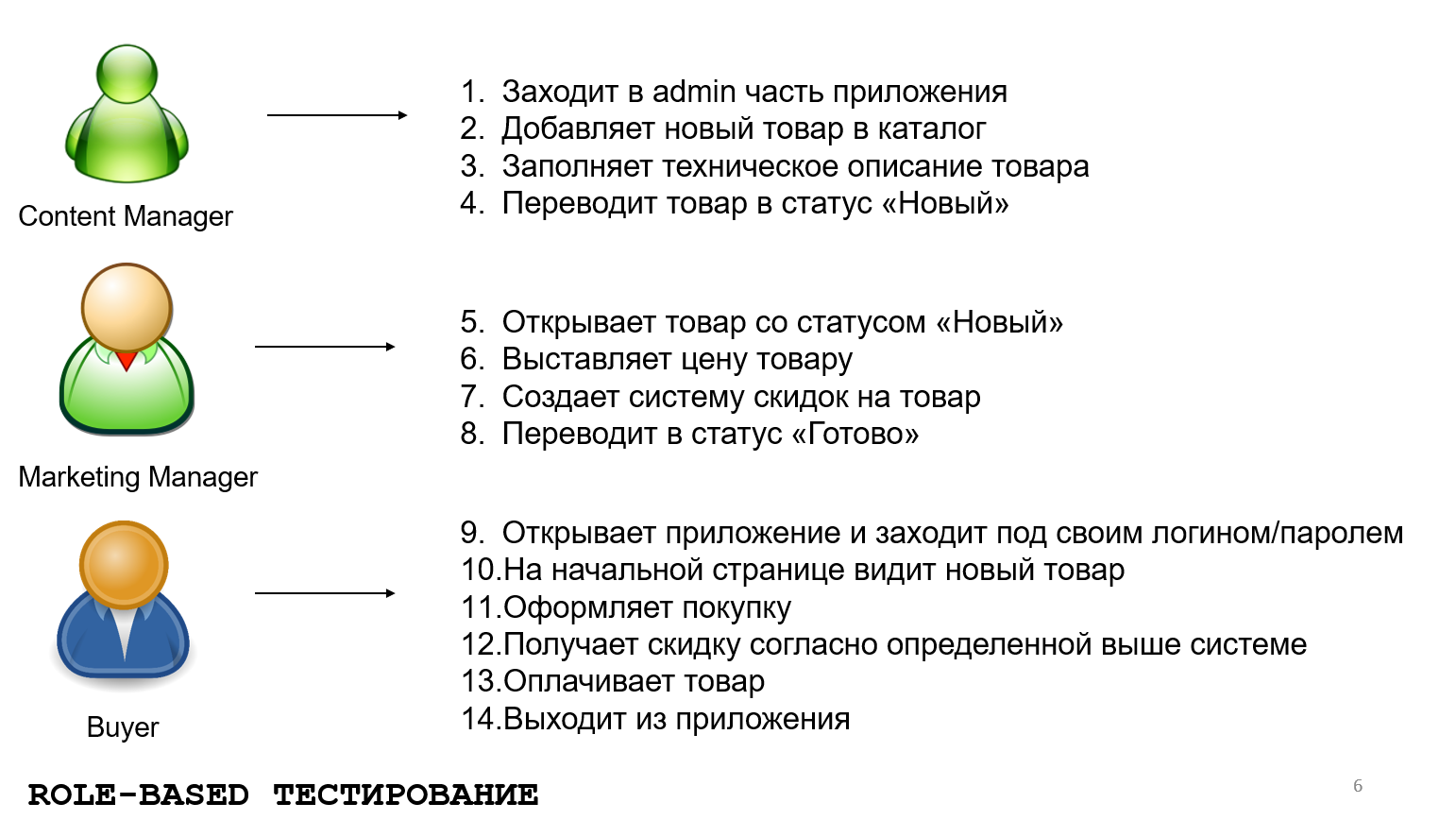


**Цели:**

Подготовиться к проведению следующих видов тестирования:

* End-to-end;
* Exploratory testing;
* Role-based testing.





**Атрибуты:**

* Идентификатор (ID);
* Краткое название (Summary, Title);
* Описание сценария (Description);
* Участники (Primary Actors);
* Предусловия (Precondition);
* Основной сценарий (Basic Flow);
* Альтернативный сценарий (Alternative Flow);
* Исключения (Exceptional Flow);
* Статус (Status);
* Приоритет (Priority);

**Статусы:**

* Не пройден (Not run);
* Успешно пройден (Passed);
* Неудачно пройденный (Failed);
* Заблокирован (Blocked);

1. **Тест план, его назначение и структура**

Тест план – 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.

**ОСНОВНАЯ ДОКУМЕНТАЦИЯ ПРОЕКТА**

* Политика тестирования
* Стратегия тестирования
* Главный план
* Уровневый план

**ПОЛИТИКА ТЕСТИРОВАНИЯ**

Политика тестирования описывает, почему организация занимается тестированием. Она определяет общие цели тестирования, которые организация хочет достигнуть

* Обобщает ценности, полученные от тестирования
* Определяет цели
* Описывает метрики
* Обрисовывает в общих чертах процесс тестирования
* Определяет действия по улучшению процесса тестирования

**СТРАТЕГИЯ ТЕСТИРОВАНИЯ**

Описывает корпоративную методологию тестирования, включая способы, которые используют при тестировании на разных уровнях, и высокоуровневые активности, связанные с тестированием.

* Стратегии, основанные на требованиях
* Стратегии, основанные на модели
* Методические стратегии
* Стратегии, совместимые с процессом, или стандартом
* Реактивные стратегии
* Консультативные стратегии
* Анти-регрессионные стратегии
* Процедуры интеграции
* Методы разработки и спецификации тестов
* Стандарты
* Тестовое окружение
* Процессы автоматизации
* Инструменты
* Роли и ответственность
* И т.д.

**ГЛАВНЫЙ ПЛАН ТЕСТИРОВАНИЯ**

Охватывает все работы по тестированию, выполняемые в конкретном проекте.

* Тестируемые и нетестируемые элементы и характеристики качества
* Программу тестирования и бюджет
* Циклы выполнения и их связь с планом выпуска ПО
* Взаимодействие отделов тестировщиков
* Критерии входа, продолжения и завершения
* Риски проекта
* Общее управление
* Входные и выходные данные

**УРОВНЕВЫЙ ПЛАН ТЕСТИРОВАНИЯ**

Описывают определенные активности, выполняемые в рамках каждого уровня тестирования, или, в некоторых случаях, типа тестирования.

В общих чертах, являются более конкретизированной версией главного плана тестирования.

Для Agile проектов планы тестирования спринта или итерации могут занять место уровневых планов тестирования.

**УПРАВЛЕНИЕ РИСКАМИ**

* Подготовка тестового окружения
* Предварительное тестирование окружения
* Тестирование ранних версий продукта
* Определение критериев входа
* Обеспечение тестируемости требований
* Рецензирование ранних версий продукта
* Участие в управлении изменениями
* Мониторинг хода выполнения проекта и его качество

1. **Отчеты об ошибках (Цель, Жизненный цикл, Атрибуты)**

Отчеты об ошибках – документ, описывающий и приоритизирующий обнаруженный дефект, а также содействующий его устранению.

Дефект – отклонение фактического результата от ожиданий наблюдателя, сформированных на основе требований, спецификаций, иной документации или опыта и здравого смысла.

**Цель:**

* **Предоставить информацию о проблеме:**

уведомить проектную команду и иных заинтересованных лиц о наличии проблемы, описать суть проблемы;

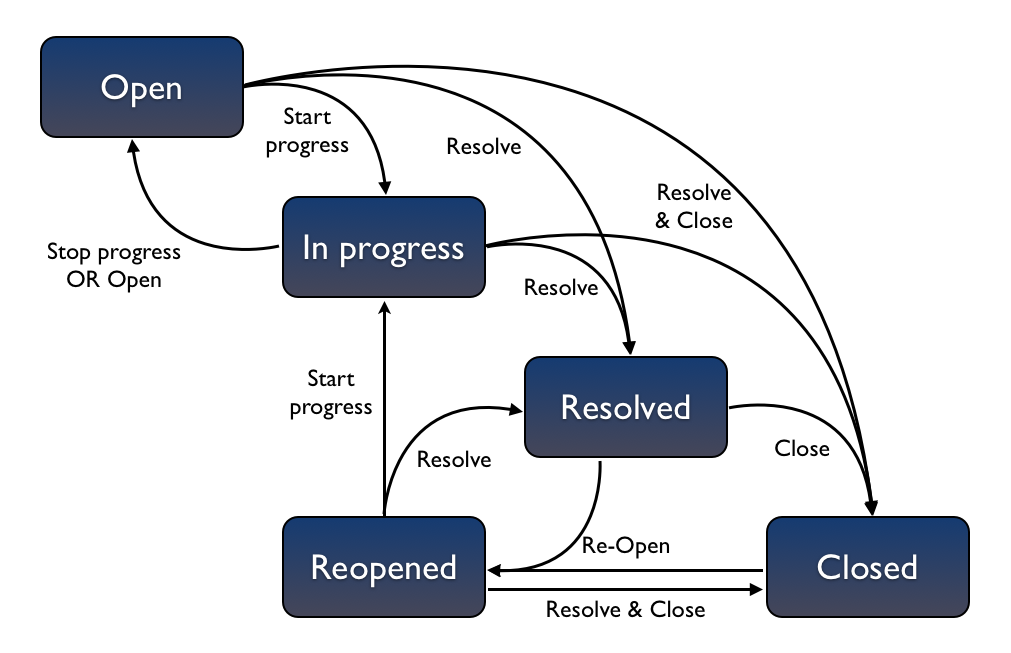
* **Приоритизировать проблему:**

определить степень опасности и желаемые сроки её устранения;

* **Содействовать устранению проблемы:**

предоставить необходимые подробности для понимания сути случившегося, а также анализ причин возникновения проблемы и рекомендации по исправлению ситуации.

**Жизненный цикл:**



Атрибуты:

* Идентификатор (ID);
* Краткое описание (Summary, Title)
* Подробное описание (Description):
  + Шаги для воспроизведения (Steps to reproduce);
  + Актуальный результат (Actual result);
  + Ожидаемый результат (Expected result);
* Воспроизводимость (Repeatability);
* Важность (Severity);
* Срочность (Priority);
* Возможность обойти (Workaround);
* Комментарий (Comment, Note);
* Среда для воспроизведения бага (Environment);
* Компонент (Component);
* Тег (Tag, Label);
* Ответственный (Assignee);
* Приложение к багу (Attachment);
* Срок исправления бага (Due Date);
* Версия билда, где был обнаружен баг (Affects Version);
* Версия, в которой должен быть исправлен баг (Fix Version);
* Создатель отчета об ошибке (Reporter);
* Оценка трудозатрат на исправление бага (Original Estimate).

1. **Отчеты об ошибках (Свойства качественных отчетов об ошибках, Типичные ошибки при написании)**

**Свойства качественных отчетов:**

* Тщательное заполнение всех полей точной и корректной информацией;
* Отсутствие «размытых» формулировок («иногда неправильно работает», «UI сломан», «кнопка Export не работает»);
* Вся информация, особенно Ожидаемый результат, корректна;
* Баг заведен на функциональность, которая разработана (неразработанный функционал за ошибку не считается);
* Отсутствует жаргонная лексика («Фигово подцепились чартники»);
* Отсутствие критики кого-то из участников команды («Ну каким дураком надо быть, чтобы такое сделать?!»);
* Важность и приоритет бага корректно выставлены;
* К отчету прикреплены все возможные документы, которые облегчат понимание причины дефекта;
* Отсутствуют грамматические и семантические ошибки («Not keyboard in parameters accepting values»);
* Использован правильный технический язык;
* Краткое описание бага должно следовать правилу WWW;
* В отчете должен быть описан только 1 дефект;
* Оформление соответствует соглашении о названиях, принятом на проекте.

**Типичные ошибки:**

* Ошибочные описания и формулировки;
* Идентичные краткое и подробное описание;
* Отсутствие фактического и ожидаемого результатов;
* Лишние пункты при воспроизведении;
* Копии экрана целиком, без отмеченной проблемы и хранящиеся на сторонних сервисах;
* Откладывание отчета на потом;
* Выдуманные ошибки;
* «не фича, а баг»;
* Заниженные (завышенные) срочность и важность;
* Концентрация на мелочах;
* Указание в шагах воспроизведения неважной информации;
* Отсутствие в шагах воспроизведения важной информации;
* Игнорирование последовательных багов.

1. **Отчеты о результатах тестирования**

Отчет о результатах тестирования – официальный документ, описывающий результаты, полученные в ходе проведения определенных видов/уровней тестирования.

**Цель:**

* Предоставить статистику по количеству проверок и результатам их выполнения;
* Предоставить статистику по количеству и серьезности найденных дефектов в приложении;
* Повысить осведомленность команды о качестве текущей версии приложения;
* Обратить внимание команды и владельцев бизнеса на области, которые нуждаются в исправлении дефектов
* Дать рекомендации относительно выпуска или наоборот задержки выпуска версии в релиз.

**Структура:**

1. Уровень/вид проведенного тестирования;
2. Количество пройденных проверок (test cases/test scenarios/checkpoints):
   * Общее количество;
   * Количество успешно пройденных (passed);
   * Количество неудачных (failed);
   * Количество заблокированных проверок (blocked);
3. Количество найденных багов:
   * Общее количество
   * Количество critical багов
   * Количество major багов
   * Количество minor багов
   * Количество trivial багов
4. Области / компоненты, которые наиболее повреждены багами (больше 75% проверок failed)
5. Проблемы, с которыми столкнулись в процессе выполнения задания.
6. **Автоматизация тестирования (Виды, Модульное тестирование)**

Автоматизированное тестирование – часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Оно использует программные средства для выполнения тестов и проверки результатов выполнения, что помогает сократить время тестирования и упростить его процесс.

Уровни:

* Уровень модульного тестирования (unit tests);
* Уровень функционального тестирование (non-UI tests);
* Уровень тестирования через пользовательский интерфейс (UI tests).

Модульное тестирование – Процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Правила Unit-тестов:

* достоверными;
* не зависеть от окружения, на котором они выполняются;
* легко поддерживаться;
* легко читаться и быть простыми для понимания (даже новый разработчик должен понять что именно тестируется);
* соблюдать единую конвенцию именования;
* запускаться регулярно в автоматическом режиме.
* Один тест должен проверять только одну сущность;
* Тесты должны храниться в системе контроля версий;
* Названия методов должны быть «говорящими»
* Необходимо придерживаться единого стиля написания тела теста
* Использование уже готовых фреймворков

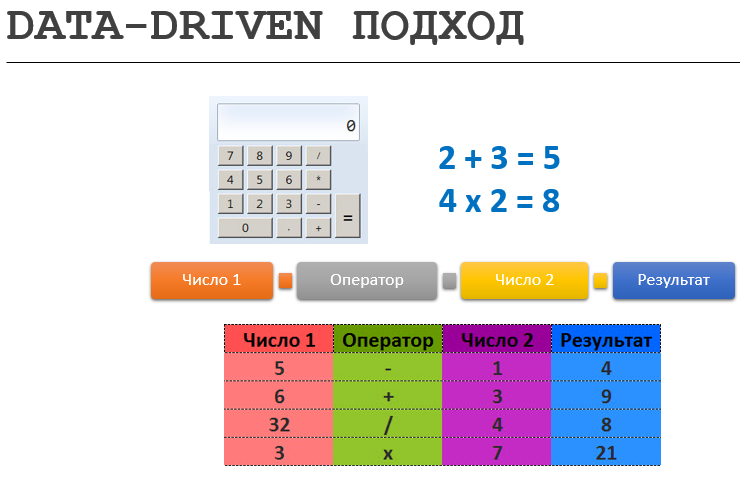
1. **Автоматизация тестирования (Интеграционные тесты, UI тестирование)**

Интеграционные тесты - это вид тестирования программного обеспечения, направленный на проверку взаимодействия между различными компонентами системы. Целью интеграционного тестирования является обнаружение ошибок в процессе интеграции различных модулей или компонентов приложения.

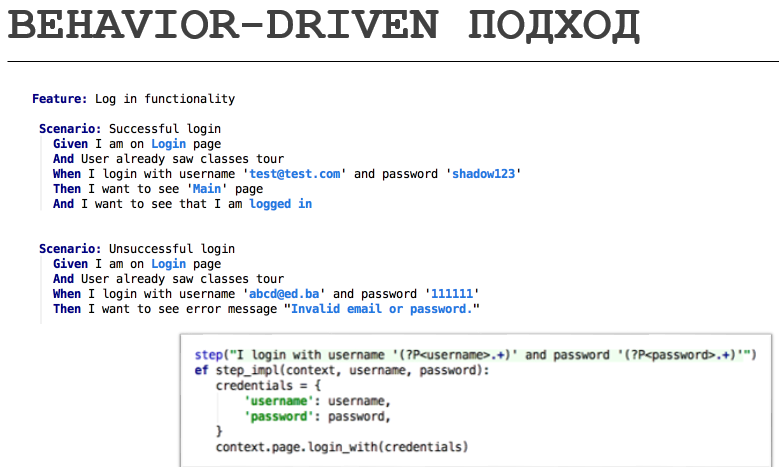
UI-тестирование – Процесс в программировании, позволяющий проверить работоспособность и внешний вид приложения используя графический интерфейс приложения.

**ПОДХОДЫ В UI АВТОМАТИЗАЦИИ**

* Data-driven подход;
* Keyword-driven подход;
* Behavior-driven подход.







**ОБЛАСТИ ДЛЯ АВТОМАТИЗАЦИИ**

* Труднодоступные места в системе;
* Часто используемая функциональность, риски от ошибок в которой достаточно высоки;
* Рутинные операции;
* Длинные end-to-end сценарии;
* Проверка данных, требующих точных математических расчетов.

**ТЕРМИНОЛОГИЯ**

**FEATURE** - функциональная возможность необычное дизайнерское решение свойство приложения

**BUG, DEFECT, ISSUE** - ошибка, недочёт, промах, упущение и другие отклонения от нормы, из-за которых программа не работает так, как задумано, и/или выдаёт неправильный результат.

**CHANGE REQUEST** - запрос на изменение функционала или добавление фичи.

**ENHANCEMENT, IMPROVEMENT** - запрос/предложение об улучшении существующего функционала или добавление новой фичи.

**FUNCTIONAL SPECIFICATION** - документ с описанием требований ко внешнему виду и работе приложения.

**REQUIREMENT** - требование к приложению описание юзер-интерфейса описание функционала приложения.

**DEV ENVIRONMENT** - среда, где разработчик(и) отслеживают, проверяют свои изменения.

**QA ENVIRONMENT** - среда, где тестировщики проверяют качество приложения.

**STAGING ENVIRONMENT** - среда, где имитируется работа приложения, как на production environment.

**PRODUCTION ENVIRONMENT** - среда, где работают конечные пользователи приложения.

**FEATURE FREEZE** - стабилизационная фаза процесса разработки запрет на добавление дополнительных свойств запрет на внесение каких-либо изменений, содержащих в себе новые возможности.

**CODE FREEZE** - запрет на изменение кода запрет на исправление багов (исправление только критических дефектов).

**VERIFICATION** - процесс тестирования приложения на соответствие предъявленным требованиям осуществляется на протяжении всей разработки приложения.

**VALIDATION** - проверка приложения на соответствие бизнес-целям проводится, как правило, перед релизом.

**Модель разработки ПО** - структура, систематизирующая различные виды проектной деятельности, их взаимодействие и последовательность в процессе разработки ПО. Выбор той или иной модели зависит от масштаба и сложности проекта, предметной области, доступных ресурсов и множества других факторов.

**Жизненный цикл ПО** - период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.