Перейдем к основным понятиям

**Тестирование программного обеспечения (Software Testing)** — проверка соответствия реальных и ожидаемых результатов поведения программы, проводимая на конечном наборе тестов, выбранном определённым образом.  
  
**Цель тестирования** — проверка соответствия ПО предъявляемым требованиям, обеспечение уверенности в качестве ПО, поиск очевидных ошибок в программном обеспечении, которые должны быть выявлены до того, как их обнаружат пользователи программы.  
  
**Для чего проводится тестирование ПО?**

* Для проверки соответствия требованиям.
* Для обнаружение проблем на более ранних этапах разработки и предотвращение повышения стоимости продукта.
* Обнаружение вариантов использования, которые не были предусмотрены при разработке. А также взгляд на продукт со стороны пользователя.
* Повышение лояльности к компании и продукту, т.к. любой обнаруженный дефект негативно влияет на доверие пользователей.

Принципы тестирования

* **Принцип 1 — Тестирование демонстрирует наличие дефектов (Testing shows presence of defects)**.  
  Тестирование только снижает вероятность наличия дефектов, которые находятся в программном обеспечении, но не гарантирует их отсутствия.
* **Принцип 2 — Исчерпывающее тестирование невозможно (Exhaustive testing is impossible)**.  
  Полное тестирование с использованием всех входных комбинаций данных, результатов и предусловий физически невыполнимо (исключение — тривиальные случаи).
* **Принцип 3 — Раннее тестирование (Early testing)**.  
  Следует начинать тестирование на ранних стадиях жизненного цикла разработки ПО, чтобы найти дефекты как можно раньше.
* **Принцип 4 — Скопление дефектов (Defects clustering)**.  
  Большая часть дефектов находится в ограниченном количестве модулей.
* **Принцип 5 — Парадокс пестицида (Pesticide paradox)**.  
  Если повторять те же тестовые сценарии снова и снова, в какой-то момент этот набор тестов перестанет выявлять новые дефекты.
* **Принцип 6 — Тестирование зависит от контекста (Testing is context depending)**. Тестирование проводится по-разному в зависимости от контекста. Например, программное обеспечение, в котором критически важна безопасность, тестируется иначе, чем новостной портал.
* **Принцип 7 — Заблуждение об отсутствии ошибок (Absence-of-errors fallacy)**. Отсутствие найденных дефектов при тестировании не всегда означает готовность продукта к релизу. Система должна быть удобна пользователю в использовании и удовлетворять его ожиданиям и потребностям.

**Обеспечение качества (QA — Quality Assurance)** и **контроль качества (QC — Quality Control)** — эти термины похожи на взаимозаменяемые, но разница между обеспечением качества и контролем качества все-таки есть, хоть на практике процессы и имеют некоторую схожесть.  
  
**QC** (Quality Control) — Контроль качества продукта — анализ результатов тестирования и качества новых версий выпускаемого продукта.  
  
К задачам контроля качества относятся:

* проверка готовности ПО к релизу;
* проверка соответствия требований и качества данного проекта.

**QA** (Quality Assurance) — Обеспечение качества продукта — изучение возможностей по изменению и улучшению процесса разработки, улучшению коммуникаций в команде, где тестирование является только одним из аспектов обеспечения качества.  
  
К задачам обеспечения качества относятся:

* проверка технических характеристик и требований к ПО;
* оценка рисков;
* планирование задач для улучшения качества продукции;
* подготовка документации, тестового окружения и данных;
* тестирование;
* анализ результатов тестирования, а также составление отчетов и других документов.

**Верификация и валидация** — два понятия тесно связаны с процессами тестирования и обеспечения качества. К сожалению, их часто путают, хотя отличия между ними достаточно существенны.  
  
**Верификация (verification)** — это процесс оценки системы, чтобы понять, удовлетворяют ли результаты текущего этапа разработки условиям, которые были сформулированы в его начале.  
  
**Валидация (validation)** — это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, его требованиям к системе.  
  
Пример: когда разрабатывали аэробус А310, то надо было сделать так, чтобы закрылки вставали в положение «торможение», когда шасси коснулись земли. Запрограммировали так, что когда шасси начинают крутиться, то закрылки ставим в положение «торможение». Но вот во время испытаний в Варшаве самолет выкатился за пределы полосы, так как была мокрая поверхность. Он проскользил, только потом был крутящий момент и они, закрылки, открылись. С точки зрения «верификации» — программа сработала, с точки зрения «валидации» — нет. Поэтому код изменили так, чтобы в момент изменения давления в шинах открывались закрылки.  
  
Документацию, которая используется на проектах по разработке ПО, можно условно разделить на две группы:

1. Проектная документация — включает в себя всё, что относится к проекту в целом.
2. Продуктовая документация — часть проектной документации, выделяемая отдельно, которая относится непосредственно к разрабатываемому приложению или системе.

**Этапы тестирования:**

1. Анализ продукта
2. Работа с требованиями
3. Разработка стратегии тестирования и планирование процедур контроля качества
4. Создание тестовой документации
5. Тестирование прототипа
6. Основное тестирование
7. Стабилизация
8. Эксплуатация

**Стадии разработки ПО** — этапы, которые проходят команды разработчиков ПО, прежде чем программа станет доступной для широкого круга пользователей.  
  
**Программный продукт проходит следующие стадии:**

1. анализ требований к проекту;
2. проектирование;
3. реализация;
4. тестирование продукта;
5. внедрение и поддержка.

**Требования**

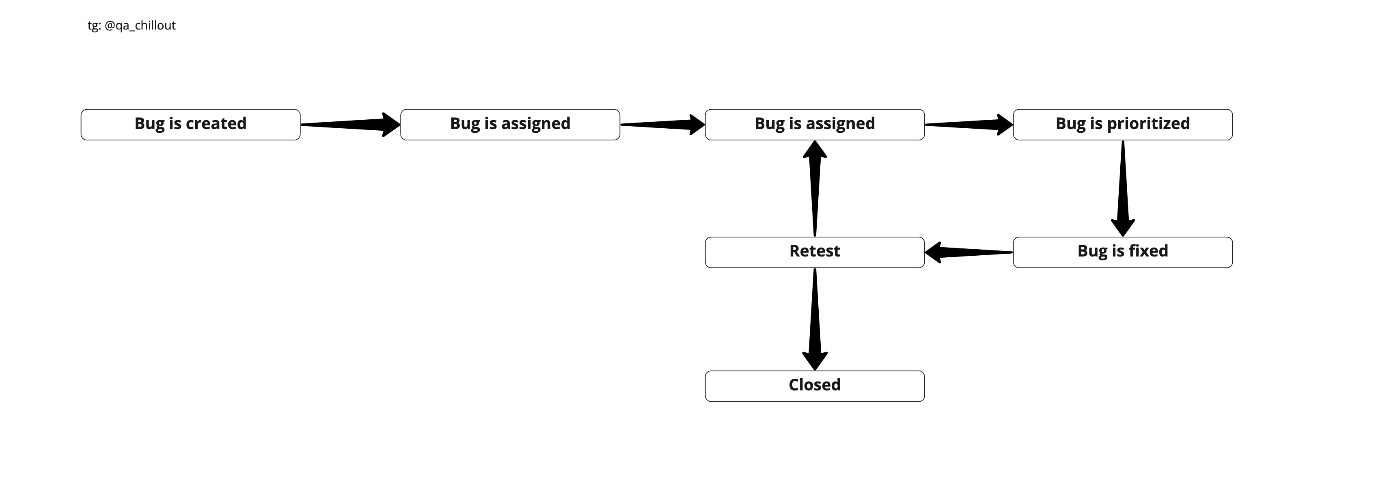
**Требования** — это спецификация (описание) того, что должно быть реализовано.  
Требования описывают то, что необходимо реализовать, без детализации технической стороны решения.  
  
**Атрибуты требований:**

1. **Корректность** — точное описание разрабатываемого функционала.
2. **Проверяемость** — формулировка требований таким образом, чтобы можно было выставить однозначный вердикт, выполнено все в соответствии с требованиями или нет.
3. **Полнота** — в требовании должна содержаться вся необходимая для реализации функциональности информация.
4. **Недвусмысленность** — требование должно содержать однозначные формулировки.
5. **Непротиворечивость** — требование не должно содержать внутренних противоречий и противоречий другим требованиям и документам.
6. **Приоритетность** — у каждого требования должен быть приоритет(количественная оценка степени значимости требования). Этот атрибут позволит грамотно управлять ресурсами на проекте.
7. **Атомарность** — требование нельзя разбить на отдельные части без потери деталей.
8. **Модифицируемость** — в каждое требование можно внести изменение.
9. **Прослеживаемость** — каждое требование должно иметь уникальный идентификатор, по которому на него можно сослаться.

**Дефект (bug)** — отклонение фактического результата от ожидаемого.  
  
**Отчёт о дефекте (bug report)** — документ, который содержит отчет о любом недостатке в компоненте или системе, который потенциально может привести компонент или систему к невозможности выполнить требуемую функцию.  
  
**Атрибуты отчета о дефекте:**

1. Уникальный идентификатор (ID) — присваивается автоматически системой при создании баг-репорта.
2. Тема (краткое описание, Summary) — кратко сформулированная суть дефекта по правилу «Что? Где? Когда?»
3. Подробное описание (Description) — более широкое описание сути дефекта (указывается опционально).
4. Шаги для воспроизведения (Steps To Reproduce) — последовательное описание действий, которые привели к выявлению дефекта. Необходимо максимально подробно расписывать, с указанием конкретных вводимых значений.
5. Фактический результат (Actual result) — указывается, что не так работает, в каком месте продукта и при каких условиях.
6. Ожидаемый результат (Expected result) — указывается, как именно должна работать система по мнению тестировщика, основанному на требованиях и прочей проектной документации.
7. Вложения (Attachments) — скриншоты, видео или лог-файлы.
8. Серьёзность дефекта (важность, Severity) — характеризует влияние дефекта на работоспособность приложения.
9. Приоритет дефекта (срочность, Priority) — указывает на очерёдность выполнения задачи или устранения дефекта.
10. Статус (Status) — определяет текущее состояние дефекта. Статусы дефектов могут быть разными в разных баг-трекинговых системах.
11. Окружение (Environment) – указывается окружение, на котором воспроизвелся баг.

**Жизненный цикл бага**

[](https://habrastorage.org/webt/dg/yy/w9/dgyyw9t6waodwooi6pnis6w2wai.jpeg)  
  
**Severity vs Priority**

**Серьёзность (severity)** показывает степень ущерба, который наносится проекту существованием дефекта. Severity выставляется тестировщиком.  
  
**Градация Серьезности дефекта (Severity):**

* **Блокирующий (S1 – Blocker)**  
  тестирование значительной части функциональности вообще недоступно. Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате которого дальнейшая работа с тестируемой системой или ее ключевыми функциями становится невозможна.
* **Критический (S2 – Critical)**  
  критическая ошибка, неправильно работающая ключевая бизнес-логика, дыра в системе безопасности, проблема, приведшая к временному падению сервера или приводящая в нерабочее состояние некоторую часть системы, то есть не работает важная часть одной какой-либо функции либо не работает значительная часть, но имеется workaround (обходной путь/другие входные точки), позволяющий продолжить тестирование.
* **Значительный (S3 – Major)**  
  не работает важная часть одной какой-либо функции/бизнес-логики, но при выполнении специфических условий, либо есть workaround, позволяющий продолжить ее тестирование либо не работает не очень значительная часть какой-либо функции. Также относится к дефектам с высокими visibility – обычно не сильно влияющие на функциональность дефекты дизайна, которые, однако, сразу бросаются в глаза.
* **Незначительный (S4 – Minor)**  
  часто ошибки GUI, которые не влияют на функциональность, но портят юзабилити или внешний вид. Также незначительные функциональные дефекты, либо которые воспроизводятся на определенном устройстве.
* **Тривиальный (S5 – Trivial)**  
  почти всегда дефекты на GUI — опечатки в тексте, несоответствие шрифта и оттенка и т.п., либо плохо воспроизводимая ошибка, не касающаяся бизнес-логики, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.

**Срочность (priority)** показывает, как быстро дефект должен быть устранён. Priority выставляется менеджером, тимлидом или заказчиком  
  
**Градация Приоритета дефекта (Priority):**

* **P1 Высокий (High)**  
  Критическая для проекта ошибка. Должна быть исправлена как можно быстрее.
* **P2 Средний (Medium)**  
  Не критичная для проекта ошибка, однако требует обязательного решения.
* **P3 Низкий (Low)**  
  Наличие данной ошибки не является критичным и не требует срочного решения. Может быть исправлена, когда у команды появится время на ее устранение.

**Существует шесть базовых типов задач:**

* **Эпик (epic)** — большая задача, на решение которой команде нужно несколько спринтов.
* **Требование (requirement )** — задача, содержащая в себе описание реализации той или иной фичи.
* **История (story)** — часть большой задачи (эпика), которую команда может решить за 1 спринт.
* **Задача (task)** — техническая задача, которую делает один из членов команды.
* **Под-задача (sub-task)** — часть истории / задачи, которая описывает минимальный объем работы члена команды.
* **Баг (bug)** — задача, которая описывает ошибку в системе.

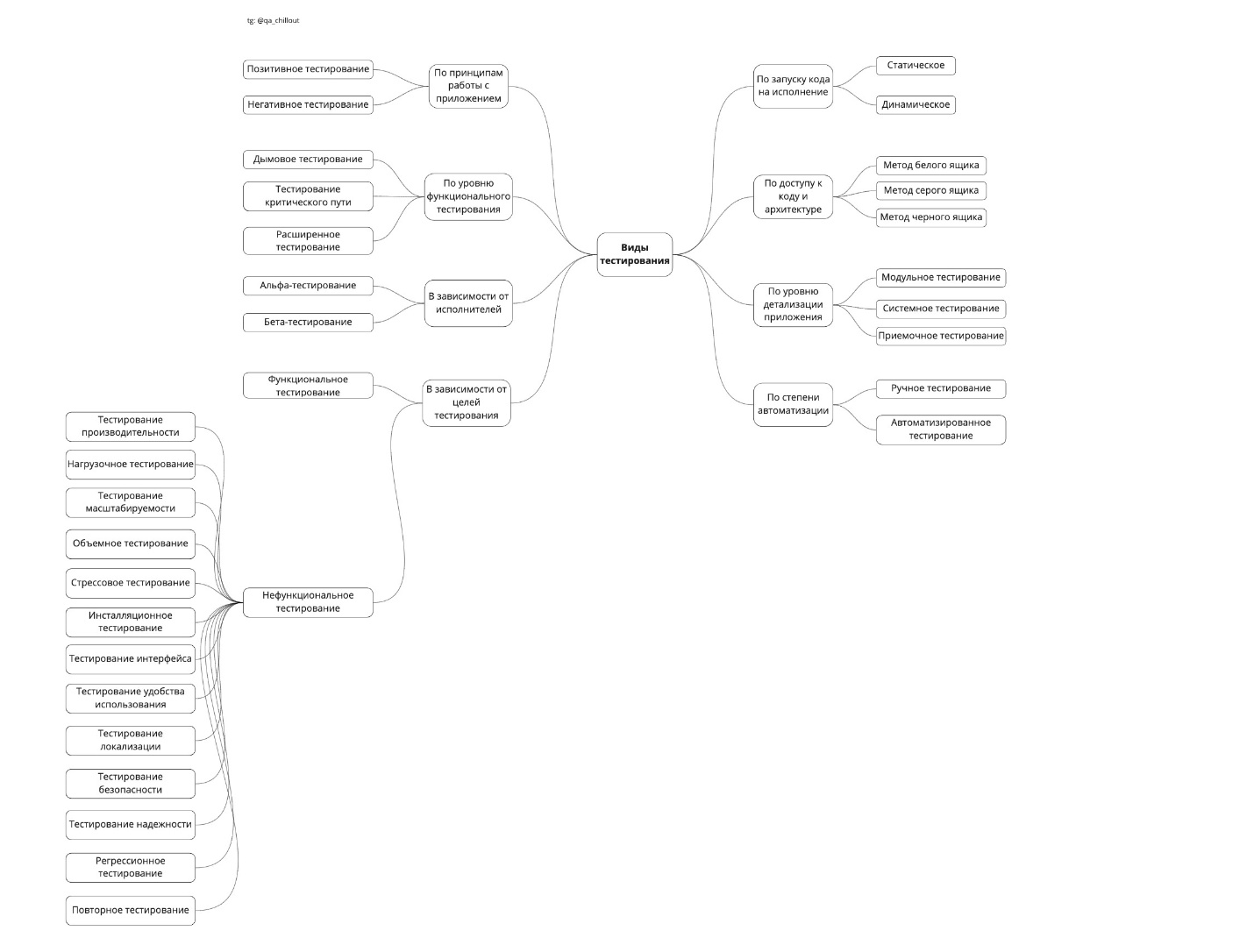
**Тестовые среды**

* **Среда разработки (Development Env)** – за данную среду отвечают разработчики, в ней они пишут код, проводят отладку, исправляют ошибки
* **Среда тестирования (Test Env)** – среда, в которой работают тестировщики (проверяют функционал, проводят smoke и регрессионные тесты, воспроизводят.
* **Интеграционная среда (Integration Env)** – среда, в которой проводят тестирование взаимодействующих друг с другом модулей, систем, продуктов.
* **Предпрод (Preprod Env)** – среда, которая максимально приближена к продакшену. Здесь проводится заключительное тестирование функционала.
* **Продакшн среда (Production Env)** – среда, в которой работают пользователи.

**Основные фазы тестирования**

* **Pre-Alpha:** прототип, в котором всё ещё присутствует много ошибок и наверняка неполный функционал. Необходим для ознакомления с будущими возможностями программ.
* **Alpha:** является ранней версией программного продукта, тестирование которой проводится внутри фирмы-разработчика.
* **Beta:** практически готовый продукт, который разработан в первую очередь для тестирования конечными пользователями.
* **Release Candidate (RC)**: возможные ошибки в каждой из фичей уже устранены и разработчики выпускают версию на которой проводится регрессионное тестирование.
* **Release:** финальная версия программы, которая готова к использованию.

**Основные виды тестирования ПО**

**Вид тестирования** — это совокупность активностей, направленных на тестирование заданных характеристик системы или её части, основанная на конкретных целях.  
  
[](https://habrastorage.org/webt/uw/-j/ul/uw-juldey34nxrwrf0kdbxztls4.jpeg)

1. **Классификация по запуску кода на исполнение:**
   * **Статическое тестирование** — процесс тестирования, который проводится для верификации практически любого артефакта разработки: программного кода компонент, требований, системных спецификаций, функциональных спецификаций, документов проектирования и архитектуры программных систем и их компонентов.
   * **Динамическое тестирование** — тестирование проводится на работающей системе, не может быть осуществлено без запуска программного кода приложения.
2. **Классификация по доступу к коду и архитектуре:**
   * **Тестирование белого ящика** — метод тестирования ПО, который предполагает полный доступ к коду проекта.
   * **Тестирование серого ящика** — метод тестирования ПО, который предполагает частичный доступ к коду проекта (комбинация White Box и Black Box методов).
   * **Тестирование чёрного ящика** — метод тестирования ПО, который не предполагает доступа (полного или частичного) к системе. Основывается на работе исключительно с внешним интерфейсом тестируемой системы.
3. **Классификация по уровню детализации приложения:**
   * **Модульное тестирование** — 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** — тестирование, направленное на проверку корректности взаимодействия нескольких модулей, объединенных в единое целое.
   * **Системное тестирование** — процесс тестирования системы, на котором проводится не только функциональное тестирование, но и оценка характеристик качества системы — ее устойчивости, надежности, безопасности и производительности.
   * **Приёмочное тестирование** — проверяет соответствие системы потребностям, требованиям и бизнес-процессам пользователя.
4. **Классификация по степени автоматизации:**
   * Ручное тестирование.
   * Автоматизированное тестирование.
5. **Классификация по принципам работы с приложением**
   * **Позитивное тестирование** — тестирование, при котором используются только корректные данные.
   * **Негативное тестирование** — тестирование приложения, при котором используются некорректные данные и выполняются некорректные операции.
6. **Классификация по уровню функционального тестирования:**
   * **Дымовое тестирование (smoke test)** — тестирование, выполняемое на новой сборке, с целью подтверждения того, что программное обеспечение стартует и выполняет основные для бизнеса функции.
   * **Тестирование критического пути (critical path)** — направлено для проверки функциональности, используемой обычными пользователями во время их повседневной деятельности.
   * **Расширенное тестирование (extended)** — направлено на исследование всей заявленной в требованиях функциональности.
7. **Классификация в зависимости от исполнителей:**
   * **Альфа-тестирование** — является ранней версией программного продукта. Может выполняться внутри организации-разработчика с возможным частичным привлечением конечных пользователей.
   * **Бета-тестирование** — программное обеспечение, выпускаемое для ограниченного количества пользователей. Главная цель — получить отзывы клиентов о продукте и внести соответствующие изменения.
8. **Классификация в зависимости от целей тестирования:**
   * **Функциональное тестирование (functional testing)** — направлено на проверку корректности работы функциональности приложения.
   * **Нефункциональное тестирование (non-functional testing)** — тестирование атрибутов компонента или системы, не относящихся к функциональности.
     1. **Тестирование производительности (performance testing)** — определение стабильности и потребления ресурсов в условиях различных сценариев использования и нагрузок.
     2. **Нагрузочное тестирование (load testing)** — определение или сбор показателей производительности и времени отклика программно-технической системы или устройства в ответ на внешний запрос с целью установления соответствия требованиям, предъявляемым к данной системе (устройству).
     3. **Тестирование масштабируемости (scalability testing)** — тестирование, которое измеряет производительность сети или системы, когда количество пользовательских запросов увеличивается или уменьшается.
     4. **Объёмное тестирование (volume testing)** — это тип тестирования программного обеспечения, которое проводится для тестирования программного приложения с определенным объемом данных.
     5. **Стрессовое тестирование (stress testing)** — тип тестирования направленный для проверки, как система обращается с нарастающей нагрузкой (количеством одновременных пользователей).
     6. **Инсталляционное тестирование (installation testing)** — тестирование, направленное на проверку успешной установки и настройки, обновления или удаления приложения.
     7. **Тестирование интерфейса (GUI/UI testing)** — проверка требований к пользовательскому интерфейсу.
     8. **Тестирование удобства использования (usability testing)** — это метод тестирования, направленный на установление степени удобства использования, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий.
     9. **Тестирование локализации (localization testing)** — проверка адаптации программного обеспечения для определенной аудитории в соответствии с ее культурными особенностями.
     10. **Тестирование безопасности (security testing)** — это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.
     11. **Тестирование надёжности (reliability testing)** — один из видов нефункционального тестирования ПО, целью которого является проверка работоспособности приложения при длительном тестировании с ожидаемым уровнем нагрузки.
     12. **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.
     13. **Повторное/подтверждающее тестирование (re-testing/confirmation testing)** — тестирование, во время которого исполняются тестовые сценарии, выявившие ошибки во время последнего запуска, для подтверждения успешности исправления этих ошибок.

**Тест-дизайн** — это этап тестирования ПО, на котором проектируются и создаются тестовые случаи (тест-кейсы).  
  
**Техники тест-дизайна**  
  
Автор книги "[A Practitioner’s Guide to Software Test Design](http://testingbooks.ru/a-practitioners-guide-to-software-test-design/)", Lee Copeland, выделяет следующие техники тест-дизайна:

1. **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.
2. **Техника анализа граничных значений (boundary value testing)** — это техника проверки поведения продукта на крайних (граничных) значениях входных данных.
3. **Попарное тестирование (pairwise testing)** — это техника формирования наборов тестовых данных из полного набора входных данных в системе, которая позволяет существенно сократить количество тест-кейсов.
4. **Тестирование на основе состояний и переходов (State-Transition Testing)** — применяется для фиксирования требований и описания дизайна приложения.
5. **Таблицы принятия решений (Decision Table Testing)** — техника тестирования, основанная на методе чёрного ящика, которая применяется для систем со сложной логикой.
6. **Доменный анализ (Domain Analysis Testing)** — это техника основана на разбиении диапазона возможных значений переменной на поддиапазоны, с последующим выбором одного или нескольких значений из каждого домена для тестирования.
7. **Сценарий использования (Use Case Testing)** — Use Case описывает сценарий взаимодействия двух и более участников (как правило — пользователя и системы).

**Методы тестирования**

**Тестирование белого ящика** — метод тестирования ПО, который предполагает, что внутренняя структура/устройство/реализация системы известны тестировщику.  
  
Согласно ISTQB, тестирование белого ящика — это:

* тестирование, основанное на анализе внутренней структуры компонента или системы;
* тест-дизайн, основанный на технике белого ящика — процедура написания или выбора тест-кейсов на основе анализа внутреннего устройства системы или компонента.
* Почему «белый ящик»? Тестируемая программа для тестировщика — прозрачный ящик, содержимое которого он прекрасно видит.

**Тестирование серого ящика** — метод тестирования ПО, который предполагает комбинацию White Box и Black Box подходов. То есть, внутреннее устройство программы нам известно лишь частично.  
  
**Тестирование чёрного ящика** — также известное как тестирование, основанное на спецификации или тестирование поведения — техника тестирования, основанная на работе исключительно с внешними интерфейсами тестируемой системы.  
  
Согласно ISTQB, тестирование черного ящика — это:

* тестирование, как функциональное, так и нефункциональное, не предполагающее знания внутреннего устройства компонента или системы;
* тест-дизайн, основанный на технике черного ящика — процедура написания или выбора тест-кейсов на основе анализа функциональной или нефункциональной спецификации компонента или системы без знания ее внутреннего устройства.

**Тестовая документация**

**Тест план (Test Plan)** — это документ, который описывает 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.  
  
**Тест план должен отвечать на следующие вопросы:**

* Что необходимо протестировать?
* Как будет проводиться тестирование?
* Когда будет проводиться тестирование?
* Критерии начала тестирования.
* Критерии окончания тестирования.

**Основные пункты тест плана:**

1. Идентификатор тест плана (Test plan identifier);
2. Введение (Introduction);
3. Объект тестирования (Test items);
4. Функции, которые будут протестированы (Features to be tested;)
5. Функции, которые не будут протестированы (Features not to be tested);
6. Тестовые подходы (Approach);
7. Критерии прохождения тестирования (Item pass/fail criteria);
8. Критерии приостановления и возобновления тестирования (Suspension criteria and resumption requirements);
9. Результаты тестирования (Test deliverables);
10. Задачи тестирования (Testing tasks);
11. Ресурсы системы (Environmental needs);
12. Обязанности (Responsibilities);
13. Роли и ответственность (Staffing and training needs);
14. Расписание (Schedule);
15. Оценка рисков (Risks and contingencies);
16. Согласования (Approvals).

**Чек-лист (check list)** — это документ, который описывает что должно быть протестировано. Чек-лист может быть абсолютно разного уровня детализации.  
  
Чаще всего чек-лист содержит только действия, без ожидаемого результата. Чек-лист менее формализован.  
  
**Тестовый сценарий (test case)** — это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.  
  
**Атрибуты тест кейса:**

* **Предусловия (PreConditions)** — список действий, которые приводят систему к состоянию пригодному для проведения основной проверки. Либо список условий, выполнение которых говорит о том, что система находится в пригодном для проведения основного теста состояния.
* **Шаги (Steps)** — список действий, переводящих систему из одного состояния в другое, для получения результата, на основании которого можно сделать вывод о удовлетворении реализации, поставленным требованиям.
* **Ожидаемый результат (Expected result)** — что по факту должны получить.

**Резюме**

Старайтесь понять определения, а не зазубривать. Если хотите узнать больше про тестирование, то можете почитать [Библию QA](https://github.com/VladislavEremeev/QA_bible). А если возникнет вопрос, всегда можете задать его нам в телеграм-канале [@qa\_chillout](https://t.me/qa_chillout).