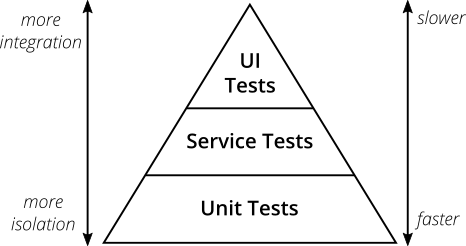
<https://habr.com/ru/post/358950/>  


*Рис. 2. Пирамида тестов*  
  
Оригинальная пирамида тестов Майка Кона состоит из трёх уровней (снизу вверх):

1. Юнит-тесты.
2. Сервисные тесты.
3. Тесты пользовательского интерфейса.
4. Писать тесты разной детализации.
5. Чем выше уровень, тем меньше тестов.

Напишите *много* маленьких и быстрых *юнит-тестов*. Напишите *несколько* более общих тестов и *совсем мало* высокоуровневых сквозных тестов, которые проверяют приложение от начала до конца.

<https://xbsoftware.ru/blog/metodologii-testirovaniya-po-kakuyu-vybrat/>

## Каскадная модель (Линейная последовательная модель жизненного цикла ПО) waterfall

Каскадная модель (Waterfall Model) является одной из наиболее старых моделей, которую можно применять не только для разработки или тестирования ПО, но также практически для любого другого проекта. Его базовым принципом является последовательный порядок выполнения задач. Это значит, что мы можем переходить к следующему шагу разработки или тестирования только после того, как предыдущий был успешно завершен. Эта модель подходит для небольших проектов и применима только в том случае, если все требования точно определены. Главными достоинствами этой методологии являются  экономическая эффективность, простота использования и управления документацией.

Процесс тестирования ПО начинается после завершения процесса разработки. На этой стадии все необходимые тесты переносятся с юнитов на системное тестирование для того, чтобы контролировать работу компонентов как по отдельности, так и в комплексе.



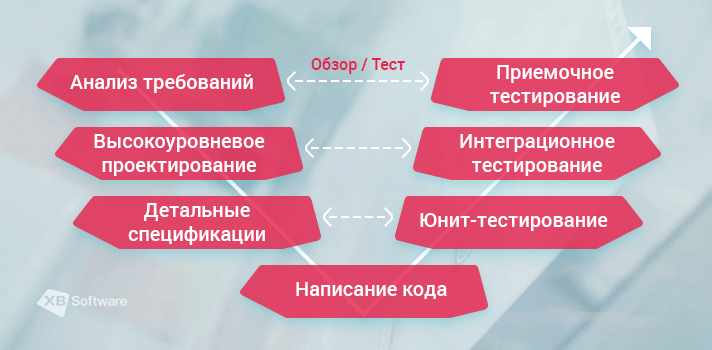
Помимо упомянутых выше достоинств, данный подход к тестированию также имеет и свои недостатки. Всегда существует вероятность обнаружения критических ошибок в процессе тестирования. Это может привести к необходимости полностью изменить один из компонентов системы или даже всю логику проекта. Но подобная задача невозможна в случае каскадной модели, поскольку возвращение на предыдущий шаг в этой методологии запрещено.

Узнайте больше о [каскадной модели](https://xbsoftware.ru/blog/zhiznennyj-tsykl-po-kaskadnaya-model-waterfall/) из предыдущей статьи.

## V-Model (Модель верификации и валидации)

Как и каскадная модель, методика V-Model основана на прямой последовательности шагов. Основным отличием между этими двумя методологиями является то, что тестирование в данном случае планируется параллельно с соответствующей стадией разработки. Согласно этой методологии тестирования ПО, процесс начинается как только определены требования и становится  возможным начать статическое тестирование, т.е. верификацию и обзор, что позволяет избежать возможных дефектов ПО на поздних стадиях. Соответствующий план тестирования создается для каждого уровня разработки ПО, что определяет ожидаемые результаты, а также критерии входа и выхода для данного продукта.

Схема данной модели показывает принцип разделения задач на две части. Те, которые относятся к дизайну и разработке, размещены слева. Задачи, относящиеся к тестированию ПО, размещены справа:



Основные этапы этой методологии могут изменяться, однако обычно они включают следующие:

* Этап **определения требований**. Приемочное тестирование относится к этому этапу. Его основная задача состоит в оценке готовности системы к финальному использованию
* Этап, на котором происходит **высокоуровневое проектирование, или High-Level Design (HDL)**. Этот этап относится к системному тестированию и включает оценку соблюдения требований к интегрированным системам
* **Фаза детального дизайна** (Detailed Design) параллельна фазе интеграционного тестирования, во время которой происходит проверка взаимодействий между различными компонентами системы
* После **этапа написания кода** начинается другой важный шаг — юнит-тестирование. Очень важно убедиться в том, что поведение отдельных частей и компонентов ПО корректно и соответствует требованиям

Единственным недостатком рассмотренной методологии тестирования является отсутствие готовых решений, которые можно было бы применить, чтобы избавиться от дефектов ПО, обнаруженных на этапе тестирования.

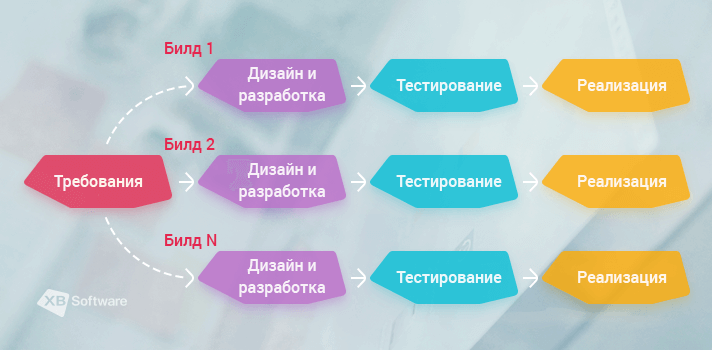
## Инкрементная модель

Данная методология может быть описана, как мультикаскадная модель тестирования ПО. Рабочий процесс разделяется на некоторое количество циклов, каждый из которых также делится на модули. Каждая итерация добавляет определенный функционал к ПО. Инкремент состоит из трех циклов:

1. **дизайн и разработка**
2. **тестирование**
3. **реализация.**

В этой модели  возможна одновременная разработка разных версий продукта. Например, первая версия может проходить этап тестирования в то время, как вторая версия находится на стадии разработки. Третья версия в то же самое время может проходить этап дизайна. Этот процесс может продолжаться до самого завершения проекта.

Очевидно, что данная методология требует обнаружения максимально возможного количества ошибок в тестируемом ПО настолько быстро, насколько это возможно. Так же, как и фаза реализации, которая требует подтверждения готовности продукта к доставке к конечному пользователю. Все эти факторы существенно увеличивают весомость требований к тестированию.



В сравнении с предыдущими методологиями, инкрементная модель имеет несколько важных преимуществ. Она более гибкая, изменение требований ведет к меньшим затратам, а процесс тестирования ПО является более эффективным, поскольку гораздо проще проводить тестирование и дебаггинг за счет использования небольших итераций. Тем не менее, стоит отметить, что общая стоимость все же выше, чем в случае каскадной модели.

## Спиральная модель (для больших проектов)

Спиральная модель это методология тестирования ПО, которая основана на инкрементном подходе и прототипировании. Она состоит из четырех этапов:

1. **Планирование**
2. **Анализ рисков**
3. **Разработка**
4. **Оценка**

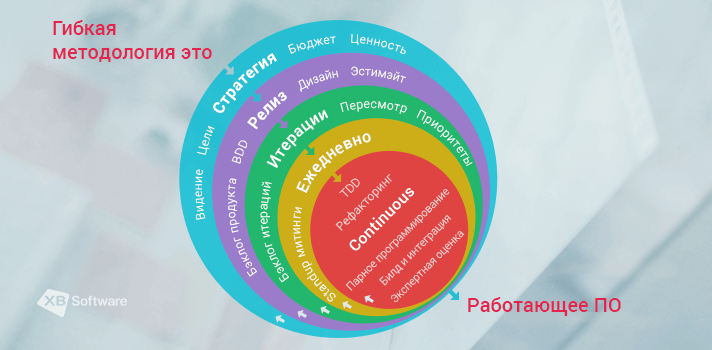
Сразу после того, как первый цикл завершен, начинается второй. Тестирование ПО начинается еще на этапе планирования и длится до стадии оценки. Основным преимуществом спиральное модели является то, что первые результаты тестирования появляется незамедлительно после появления результатов тестов на третьем этапе каждого цикла, что помогает гарантировать корректную оценку качества. Тем не менее, важно помнить о том, что эта модель может быть довольно затратной и не подходит для маленьких проектов.



Несмотря на то, что эта модель является довольно старой, она остается полезной как для тестирования, так и для разработки. Более того, главная цель многих методологий тестирования ПО, включая спиральную модель, изменилась в последнее время. Мы используем их не только для поиска дефектов в приложениях, но также и для выяснения причин, их вызвавших. Такой подход помогает разработчикам работать более эффективно и быстро устранять ошибки.

## Agile

Методология гибкой (Agile) разработки и тестирование ПО может быть описана как набор подходов, ориентированных на использование интерактивной разработки, динамического формирования требований и обеспечения их осуществления как результата постоянного взаимодействия внутри самоорганизующейся рабочей группы. Большинство гибких методологий разработки ПО нацелены на минимизацию рисков посредством разработки в рамках коротких итераций. Одним из главных принципов этой гибкой стратегии является возможность быстрого реагирования на возможные изменения, нежели стремление положиться на долгосрочное планирование.



## Экстремальное программирование (XP, Extreme Programming)

[Экстремальное программирование](https://habrahabr.ru/post/197760/" \t "https://xbsoftware.ru/blog/metodologii-testirovaniya-po-kakuyu-vybrat/_blank) является одним их примеров гибкой разработки ПО. Отличительной особенностью этой методологии является “парное программирование”, ситуация, когда один разработчик работает над кодом, в то время как его коллега постоянно проводит обзор написанного кода. Процесс тестирования ПО является довольно важным, поскольку начинается даже раньше, чем написана первая строка кода. Каждый модуль приложения должен иметь юнит-тест, чтобы большинство ошибок могло быть исправлено на стадии написания кода. Другим отличительным свойством является то, что тест определяет код, а не наоборот. Это значит, что определенная часть кода может быть признана завершенной только в том случае, если все тесты пройдены успешно. В противном случае, код отклоняется.

Главными достоинствами такой методологии являются постоянное тестирование и короткие релизы, что помогает обеспечить высокое качество кода.

## Scrum

Scrum — Часть методологии Agile, итеративный инкрементный фреймворк, созданный для управления процессом разработки ПО. Согласно принципам Scrum, команда тестировщиков должна участвовать в следующих этапах:

* Участие в Scrum планировании
* Поддержка в юнит-тестировании
* Тестирование пользовательских историй
* Сотрудничество с заказчиком и владельцем продукта для определения критериев приемлемости
* Предоставление автоматического тестировании

Более того, участники QA-отдела должны присутствовать на всех ежедневных собраниях, как и другие члены команды, чтобы обсудить, что было протестировано и сделано вчера, что будет протестировано сегодня, а также общий прогресс тестирования.

В то же время принципы Agile методологии в Scrum к появлению специфических особенностей:

* Оценка усилий, необходимых для каждой пользовательской истории является обязательной
* Тестировщик должен быть внимательным к требованиям, поскольку они могут постоянно изменяться
* Риск регрессии возрастает вместе с частыми изменениями в коде
* Одновременность планирования и выполнения тестов
* Недопонимание между членами команды в случае если требования заказчика не до конца ясны

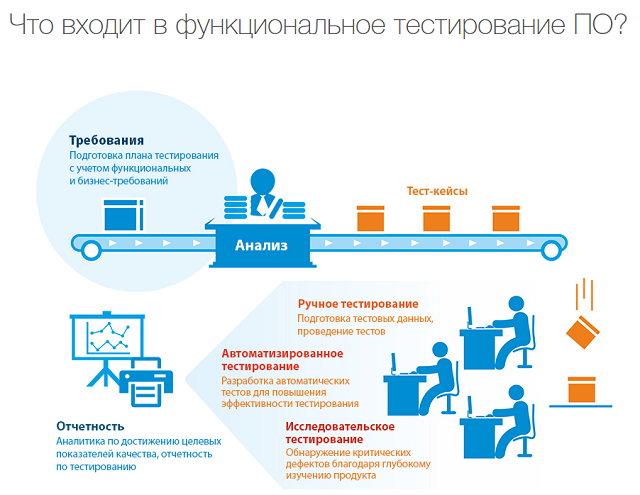
Узнайте больше о [методологии Scrum](https://xbsoftware.ru/blog/zhiznennyj-tsykl-po-skram-po-shagam/) из предыдущей статьи.

## **1. Виды тестирования по целям**

В зависимости от того, какие цели вы преследуете, тестируя ту или иную программу, тестирование бывает:

* ****Функциональное.****
* ****Нефункциональное.****

Функциональное тестирование направлено на проверку того, какие функции ПО реализованы, и того, насколько верно они реализованы. = соответствие требованиям



Нефункциональное – проверяет корректность работы нефункциональных требований. Этот вид тестирования скорее проверяет, КАК программный продукт работает. Он включает в себя следующие виды:

* ****Тестирование производительности**** – проверяет как ПО работает под определенной нагрузкой.
* ****Тестирование пользовательского интерфейса**** – определяет удобство пользования разными параметрами интерфейса (кнопки, цвета, выравнивание и т. д.).
* ****Тестирование удобства использования**** – проверяет, удобен ли ПО в использовании.
* ****Тестирование защищенности**** – определяет, насколько безопасно использование программного продукта: защищено ли ПО от атак хакеров, несанкционированного доступа к данным и т. д.
* ****Инсталляционное тестирование**** – проверяет, не возникает ли проблем при установке, удалении, а также обновлении ПО.
* ****Тестирование совместимости**** – тестирование работы программного продукта в определенном окружении.
* ****Тестирование надежности**** – проверяет работу ПО при длительной средней ожидаемой нагрузке.
* ****Тестирование локализации**** – тестирование локализованной версии программного продукта (языковой и культурный аспекты).

## **2. По степени автоматизации**

В зависимости от того, используют ли тестировщики дополнительные программные средства для тестирования приложений или программ, тестирование бывает:

* ****Ручное**** – без использования дополнительных программных средств, т. е. тестирование «вручную».
* ****Автоматизированное**** – с использованием программных средств (более детально в описании [курса по автоматизации тестирования ПО](https://qa-academy.by/programmy-i-kursy/distancionnoe-obuchenie-avtomatizaciya-testirovaniya-po-bazovyj-kurs/" \t "https://qa-academy.by/qaacademy/news/klassifikaciya-vidov-testirovaniya/_blank)).

## **3. По позитивности сценария**

По позитивности сценария тестирование бывает:

* ****Позитивным**** – проверка ПО на соответствие ожидаемому поведению. Это самый первый вид тестирования, который следует проводить, ведь основная задача тестирования – проверить, корректно ли работает программа.
* ****Негативным**** – проверяет, будет ли ПО работать в случае, когда поведение пользователя отличается от ожидаемого.

## **4. По доступу к коду программного продукта**

В процессе тестирования инженер может работать с ПО, не обращаясь к его коду, а может определить правильность работы, взглянув на код. По доступу к коду программного продукта тестирование делится на:

* ****Тестирование «белого ящика»**** – тестирование программного продукта с доступом к коду.
* ****Тестирование «черного ящика»**** – тестирование без доступа к коду продукта.
* ****Тестирование «серого ящика»**** – тестирование, основанное на ограниченном знании внутренней структуры ПО. Часто говорят, что это смесь тестирования «белого ящика» и «черного ящика», но это в корне неверно. В данном случае тестировщик не работает с кодом программного продукта, но он знаком с внутренней структурой программы и взаимодействием между компонентами.



## **5. По уровню**

По уровню тестирование бывает:

* ****Модульное / юнит-тестирование**** – проверка корректной работы отдельных единиц ПО. Этот вид тестирования могут выполнять сами разработчики.
* ****Интеграционное тестирование**** – проверка взаимодействия между несколькими единицами ПО.
* ****Системное**** – проверка работы всей системы на соответствие заявленным требованиям к программному продукту.

## **6. По исполнителю**

Наверняка, вы слышали об альфа- и бета-тестировании. А поучаствовать в одном из них можно, даже не будучи тестировщиком. Итак, по исполнителю тестирование делится на:

* ****Альфа-тестирование**** – тестирование программного продукта на поздней стадии разработки. Проводится разработчиками или тестировщиками.
* ****Бета-тестирование**** – тестирование ПО перед выходом на рынок силами обычных людей – добровольцев, которым передается предварительная версия продукта (бета-версия). Их отзывы собираются, анализируются и учитываются при внесении правок в продукт.

## **7. По формальности**

По формальности тестирование бывает:

* ****Тестирование по тестам**** – тестирование по предварительно написанным тест-кейсам.
* ****Исследовательское тестирование**** – одновременная разработка тестов и их исполнение.
* ****Свободное тестирование**** –  тестирование без разработки тестов, без документации. Основывается на интуиции и опыте тестировщика.

## **8. По важности**

По степени важности тестируемых функций тестирование делится на:

* ****Дымовое тестирование**** – проверка самой важной функциональности программного продукта.
* ****Тестирование критического пути**** – проверка функциональности, используемой типичными пользователями в повседневной деятельности.
* ****Расширенное тестирование**** – проверка всей заявленной функциональности.

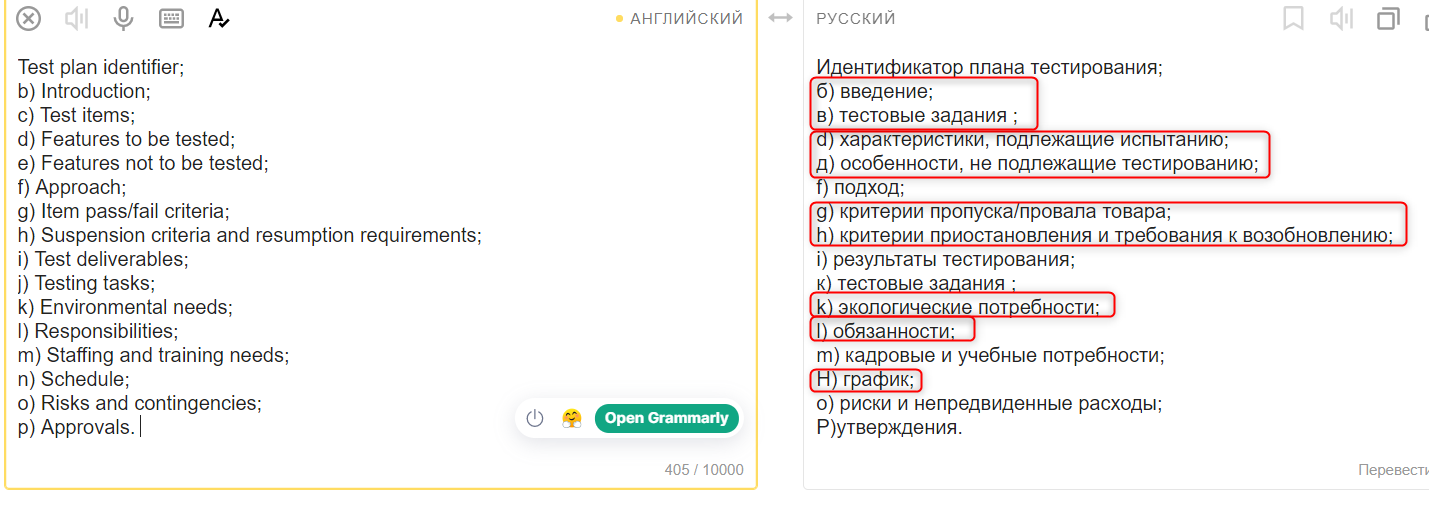
Регрессионное тестирование проводится с целью проверить, не влияют ли новые функции, улучшения и исправленные дефекты на существующую функциональность продукта и не возникают ли старые дефекты.

Каждая сложная программная система состоит из отдельных частей - модулей, выполняющих ту или иную функцию в составе системы. Для того, чтобы удостовериться в корректной работе всей системы, необходимо вначале протестировать каждый модуль системы по отдельности. В случае возникновения проблем при тестировании системы в целом это позволяет проще выявить модули, вызвавшие проблему, и устранить соответствующие дефекты в них. Такое тестирование модулей по отдельности получило называние модульного тестирования.

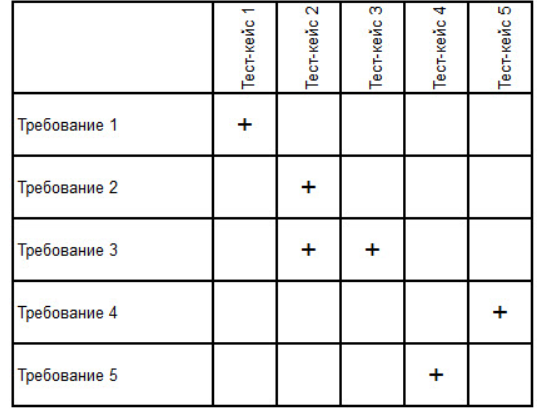
<https://habr.com/ru/post/279535/>

****Тест план (Test Plan)**** — это документ, описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.  
Отвечает на вопросы:  
Что надо тестировать?  
Что будете тестировать?  
Как будете тестировать?  
Когда будете тестировать?  
Критерии начала тестирования.  
Критерии окончания тестирования.

****Основные пункты тест плана****

********

****Traceability matrix — Матрица соответствия требований**** — это двумерная таблица, содержащая соответствие функциональных требований (functional requirements) продукта и подготовленных тестовых сценариев (test cases). На пересечении — отметка, означающая, что требование текущей колонки покрыто тестовым сценарием текущей строки.  
Матрица соответствия требований используется QA-инженерами для валидации покрытия продукта тестами. МСТ является неотъемлемой частью тест-плана.



****Тестовый сценарий (Test Case)**** — это совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

Каждый тест кейс должен иметь 3 части:  
PreConditions

Test Case Description + ожидаемые результаты

PostConditions Список действий, переводящих систему в первоначальное состояние (состояние до проведения теста — initial state)

****Чек-лист (check list)**** — это документ, описывающий что должно быть протестировано. При этом чек-лист может быть абсолютно разного уровня детализации.

Как правило, чек-лист содержит только действия (шаги), без ожидаемого результата.

****Дефект (он же баг)**** – это несоответствие фактического результата выполнения программы ожидаемому результату.

****Error**** — ошибка пользователя, то есть он пытается использовать программу иным способом.

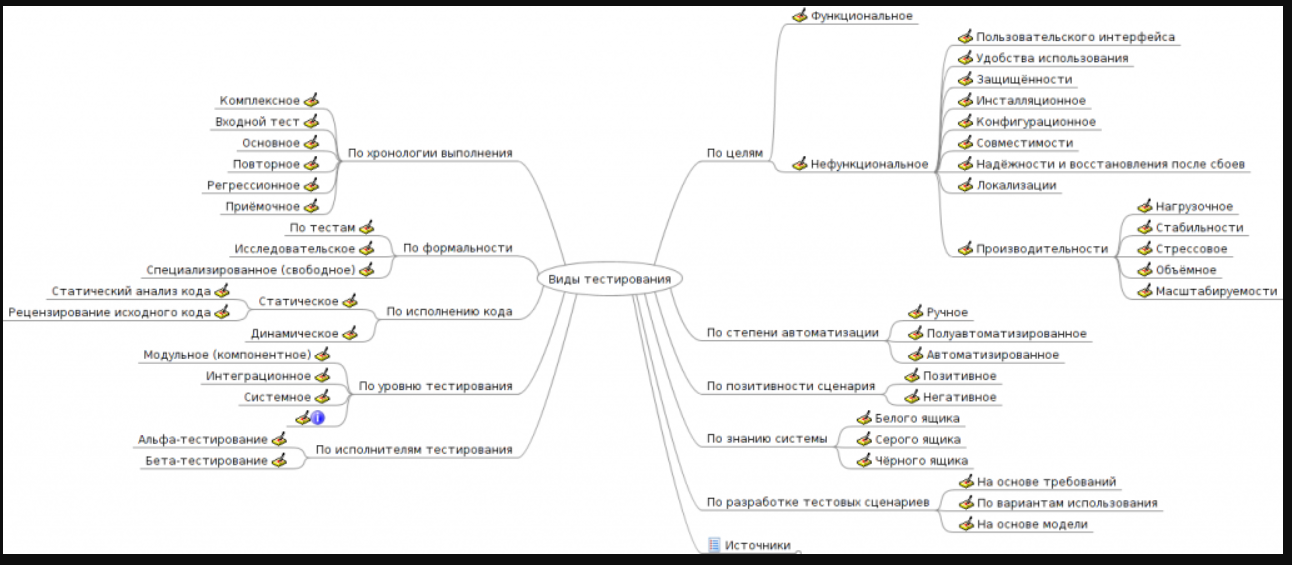
****Bug (defect)**** — ошибка программиста

****Failure**** — сбой

****Баг Репорт (Bug Report)**** — это документ, описывающий ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

**Серьезность (Severity)** Наиболее распространена пятиуровневая система градации серьезности дефекта:  
• S1 Блокирующий (Blocker)  
• S2 Критический (Critical)  
• S3 Значительный (Major)  
• S4 Незначительный (Minor)  
• S5 Тривиальный (Trivial)

****Severity vs Priority****  
Серьезность (Severity) — это атрибут, характеризующий *влияние дефекта на работоспособность приложения.*Приоритет (Priority) — это атрибут, указывающий на очередность выполнения задачи или устранения дефекта. Можно сказать, что это инструмент менеджера по планированию работ. Чем выше приоритет, тем быстрее нужно исправить дефект.  
Severity выставляется тестировщиком  
Priority – менеджером, тимлидом или заказчиком  
  
Градация Серьезности дефекта (Severity)  
  
****S1 Блокирующая (Blocker)****  
Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате которого дальнейшая работа с тестируемой системой или ее ключевыми функциями становится невозможна. Решение проблемы необходимо для дальнейшего функционирования системы.  
  
****S2 Критическая (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, без возможности решения проблемы, используя другие входные точки. Решение проблемы необходимо для дальнейшей работы с ключевыми функциями тестируемой системой.  
  
****S3 Значительная (Major)****  
Значительная ошибка, часть основной бизнес логики работает некорректно. Ошибка не критична или есть возможность для работы с тестируемой функцией, используя другие входные точки.  
  
****S4 Незначительная (Minor)****  
Незначительная ошибка, не нарушающая бизнес логику тестируемой части приложения, очевидная проблема пользовательского интерфейса.  
  
****S5 Тривиальная (Trivial)****  
Тривиальная ошибка, не касающаяся бизнес логики приложения, плохо воспроизводимая проблема, малозаметная посредствам пользовательского интерфейса, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.  
  
  
  
Градация Приоритета дефекта (Priority)  
****P1 Высокий (High)****  
Ошибка должна быть исправлена как можно быстрее, т.к. ее наличие является критической для проекта.  
****P2 Средний (Medium)****  
Ошибка должна быть исправлена, ее наличие не является критичной, но требует обязательного решения.  
****P3 Низкий (Low)****  
Ошибка должна быть исправлена, ее наличие не является критичной, и не требует срочного решения.



Зачем нужна стратегия тестирования?

-для начала неплохо бы осознать, какими ресурсами мы располагаем.

# **1, Какие типы тестирования будем применять на проекте и почему**

### Тестирование локализации

### Регрессионное тестирование

### 2,Кроссбраузерность и кроссплатформенность

# 3,. Приоритеты в тестировании

 Например, разработку автотестов стоит планировать с учетом приоритетов: в первую очередь автоматизируется проверка наиболее критичных сценариев (smoke-тестирование).

# 4. Работа тестировщика на проекте

Для кого-то в команде может стать сюрпризом, что тестировщик что-то делает или чего-то не делает.

# 5. Тестовая документация на проекте

* какую тестовую документацию будем вести на проекте,
* будем ли писать тест-кейсы,
* ведем ли чек-листы
* и как часто обновлять эту документацию.

# 8. Критерии начала и окончания тестирования

# 9. Инструменты для работы

Раздел полезен затем, чтобы уже на этапе планирования тестирования поставить задачи ответственным (админу и менеджеру) и к началу работ получить все необходимые инструменты.  
  
Стоит обсудить такие вопросы:

* Какие инструменты нам нужны для ведения тестовой документации,
* Какие инструменты нужны для автоматизации.