**Тестирование -** проверка соответствия между реальным и ожидаемым поведением программы, проводится на наборе тестов.   
**Цель тестирования** — проверка соответствия предъявляемым требованиям, обеспечение уверенности в качестве, поиск очевидных ошибок, которые должны быть выявлены до того, как их обнаружат пользователи программы. **Для чего проводится тестирование?** Для проверки соответствия требованиям.  
**Типы тестирования:  
Функциональное тестирование** проводится для проверки критически важных для бизнеса функций, функциональности и удобство использования. Функциональное тестирование гарантирует, что функции программного обеспечения и функциональные возможности ведут себя так, как ожидалось, без каких-либо сбоев. В основном проверяется все приложение на спецификации, упомянутые в документе Спецификация требований к программному обеспечению (SRS). Типы функциональных тестов включают в себя унитарное тестирование, тестирование интерфейса, регрессионные испытания, а также многие из них.

**Тестирование единицы**

Одноразовая тестирование фокусируется на тестировании отдельных частей/единиц программного приложения в начале SDLC. Любая функция, процедура, метод или модуль могут быть единицей для прохождения модульного тестирования для определения его правильности и ожидаемого поведения. Унитарное тестирование является первым тестированием, которое разработчики выполняют на этапе разработки.

**Интеграционное тестирование** включает тестирование различных модулей программного приложения в группе. Программное приложение состоит из различных подмодулей, которые работают вместе для различных функций. Целью интеграционного тестирования является проверка совместной интеграции различных модулей и выявление связанных с ними ошибок и проблем.

**Нефункциональное тестирование** подобно функциональному тестированию; однако основное отличие заключается в том, что эти функции проверяются под нагрузкой на производительность наблюдателя, надежность, удобство использования, масштабируемость и т.д. Нефункциональные испытания, такие как тестирование нагрузки и стресса, обычно проводятся с использованием средств автоматизации и решений, таких как LoadView. Помимо тестирования производительности, типы нефункционального тестирования включают тестирование установки, тестирование надежности и тестирование безопасности.

**Тестирование производительности** — это тип нефункционального тестирования, который проводится для определения скорости, стабильности и масштабируемости программного приложения. Как следует из названия, общая цель этого тестирования заключается в проверке производительности приложения в отношении различных системных и сетевых критериев, таких как использование процессора, скорость загрузки страниц, пик обработки трафика, использование серверных ресурсов и т.д. В рамках тестирования производительности существует несколько других типов тестирования, таких как тестирование нагрузки и стресс-тестирование.  
**Как эти типы тестов отличаются друг от друга**

Возможно, вы имеете некоторое представление о вышеуказанных различных типах тестирования. Все тесты сосредоточены на надежности и готовности программных приложений, однако, давайте лучше понять различия между ними на некоторых примерах. Допустим, у вас есть веб-сайт электронной коммерции / приложение со стандартными функциональными возможностями. Вот несколько примеров тестирования производительности, функционального тестирования, интеграционного тестирования и удельного тестирования:

Если вы хотите проверить, как ваш сайт будет работать, когда большое количество пользователей приходит на ваш сайт, например, во время сезона продаж, вам нужно выполнить нагрузочных тестов, которые подпадают под категорию тестирования производительности. Это поможет вам обнаружить проблемы скорости и стабильности и устранить потенциальные узкие места производительности.

Предположим, что вы хотите проверить входные данные и выход для каждой функциональности, такой как регистрация, войти, добавить в корзину, выезд, обработку платежей, записи в базу данных и т.д., в соответствии с тестовыми случаями, написанными в документе SRS. В этом случае необходимо выполнить функциональное тестирование.

Если вы хотите проверить функциональность корзины с интеграцией кассы и платежного модуля, чтобы увидеть, если количество элементов, добавленных в корзину успешно приобретены с правильной оплатой, вам нужно сделать интеграционное тестирование.

Если вы написали модуль для загрузки продукта и хотите проверить, правильно ли он и продукты успешно добавляются без каких-либо ошибок или дефектов, вам нужно сделать модуль загрузки продукта.

Подводя итог, можно сказать, что тестирование производительности проводится для проверки производительности веб-сайта. **Функциональное тестирование** проводится для проверки всех функциональных возможностей. Интеграционное тестирование проводится для проверки взаимодействия между различными модулями, а модульное тестирование проводится для проверки отдельных частей кода на правильность.

Преимущества этих типов тестов

**Тестирование производительности**

Оценивает скорость и масштабируемость веб-сайта/приложения.

Определяет узкие места для улучшения производительности.

Обнаруживает ошибки, которые упускаются из виду при функциональном тестировании.

Оптимизация системы и усовершенствования функций

Обеспечивает надежность сайта под большой нагрузкой.

Функциональное тестирование

Убедитесь, что веб-сайт/приложение не имеет дефектов.

Обеспечивает ожидаемое поведение всех функциональных возможностей.

Гарантирует правильность архитектуры при необходимой безопасности.

Улучшает общее качество и функциональность.

Минимизирует бизнес-риски, связанные с веб-сайтом/приложением.

**Интеграционное тестирование**

Гарантирует, что все модули приложений хорошо интегрированы и работают, как ожидалось вместе.

Обнаруживает взаимосвязанные проблемы и конфликты, чтобы решить их на ранней стадии, прежде чем они создадут большую проблему.

Проверяет функциональность, надежность и стабильность между различными модулями.

Обнаруживает упущенные исключения для улучшения качества кода.

Поддерживает конвейер CI/CD.

Тестирование единицы

Раннее обнаружение ошибок в недавно разработанных функциональных возможностях или функциях.

Минимизирует затраты на тестирование при раннем обнаружении проблем.

Улучшает качество кода с лучшей рефакторингом кода.

Поддерживает процесс гибкой разработки.

Упрощает интеграцию и позволяет хорошую документацию.

Недостатки этих типов тестирования

Поскольку все эти типы тестов улучшают функциональные возможности и улучшают пользовательский опыт, у них нет никаких недостатков. Единственное, что вы можете рассмотреть недостаток, в общем, это время и стоимость, связанные с тестированием. Тестирование требует усилий и ресурсов, и существует риск, связанный с неточными результатами испытаний. Тем не менее, не делать веб-сайт / тестирование приложений поставит вас в компрометирующее положение, которое может помешать вашему бизнесу и репутации значительно.

Правильное время для выполнения этих типов тестов

**Тестирование производительности** является обязательным во всех средах разработки и производства, чтобы убедиться, что ваш веб-сайт / приложение до скорости и может выдержать ожидаемую нагрузку пользователя. Функциональное тестирование должно быть сделано с каждой сборкой для проверки всех изменений и функций в соответствии со спецификациями и требованиями. Интеграционное тестирование должно быть сделано при интеграции нового фрагмента кода с другим модулем, чтобы убедиться, что нет конфликтов и работать вместе правильно. Одноразовое тестирование должно быть сделано разработчиками всякий раз, когда они делают написание любого кода для проверки правильного ввода и вывода.

Совет: руконый подход

Хотя каждый тип тестирования кажется отдельной задачей, вы можете объединить их бойко для достижения большего качества продукции. Возьмем пример.

Допустим, вы создали новую веб-страницу, запуск теста нагрузки (тест производительности) с помощью **LoadView** для этой веб-страницы в качестве удельного теста будет убедиться, что, когда вы сделаете окончательный построить со всеми страницами, веб-сайт уже оптимизирован для обработки высокой нагрузки пользователей в пиковых сценариях трафика. Это фактически означает, что тестирование производительности является частью удельного тестирования. Такой подход поможет вам сократить количество проблем на ранней стадии и сэкономить много затрат и времени в долгосрочной перспективе.

Что такое LoadView?

**LoadView** — это облачный инструмент тестирования нагрузки, который проверяет производительность веб-сайта в условиях высокого трафика. Он имитирует тысячи виртуальных пользователей из разных географических мест на нескольких браузерах и устройствах для создания наиболее реалистичных сред для тестирования производительности. Он также предлагает функцию создания тестового сценария с помощью EveryStep Web Recorder, которая не требует каких-либо навыков кодирования, чтобы любой человек в вашей команде мог создавать сценарии и выполнять тестирование нагрузки. Вы можете протестировать свой веб-сайт, приложение, веб-страницы и API сторонних разработчиков с помощью LoadView для обнаружения узких мест в производительности и быстрого их устранения. LoadView быстро приутовиться, и его отчеты о производительности легко понять с действиями идеи.

Вывод: Типы тестирования программного обеспечения

**Тестирование производительности** является обязательным во всех средах разработки и производства, чтобы убедиться, что ваш веб-сайт / приложение до скорости и может выдержать ожидаемую нагрузку пользователя. Функциональное тестирование должно быть сделано с каждой сборкой для проверки всех изменений и функций в соответствии со спецификациями и требованиями. Интеграционное тестирование должно быть сделано при интеграции нового фрагмента кода с другим модулем, чтобы убедиться, что нет конфликтов и работать вместе правильно. Одноразовое тестирование должно быть сделано разработчиками всякий раз, когда они делают написание любого кода для проверки правильного ввода и вывода.  
**Контрольный список/лист проверок/чек-лист** — это список проверок, которые помогают тестировщику протестировать приложение или отдельные функции. Основная цель чек-листа состоит в том, чтобы вы не забыли проверить всё, что планировали.  
**Баг-репорт (bug report)** — это технический документ, который подробно описывает ошибку в работе программы, приложения или другого ПО. Его составляет тестировщик, чтобы разработчикам было понятно, что работает неправильно, насколько дефект критичен и что нужно исправить.  
**Баг-репорты** — часть рабочего процесса. В них фиксируют наличие ошибки, назначают ответственного за исправление. Если сообщить об ошибке в рабочем чате, о ней скорее всего забудут. Каждый член команды подумает, что ошибку исправит другой, и в итоге она так и останется в коде.  
**Виды багов**

**Функциональные**. Возникают, когда фактический результат работы не соответствует ожиданиям: не получается опубликовать комментарий на сайте, добавить товар в корзину или открыть страницу.

**Визуальные**. Это случаи, когда приложение выглядит иначе, чем задумано: кнопка накладывается на текст, не отображаются картинки или текст выходит за пределы окна.

**Логические**. Баг, при котором что-то работает неправильно с точки зрения логики, — например, когда можно указать несуществующую дату (31 февраля) или поставить дату рождения из будущего (2077 год).

**Дефекты UX**. Приложение или программа неудобны в использовании: при просмотре ленты новостей пользователя постоянно отбрасывает к началу, слишком близко расположены кнопки и вместо одной нажимается другая.

**Дефекты безопасности.** Случаи, когда из-за ошибки в коде данные пользователей (почты, пароли, фото, информация о платежах) могут быть доступны третьим лицам.  
**Шаблон баг-репорта**

Шаблон отчета о дефекте, который отвечает запросам тестировщика и программиста, выглядит следующим образом:

1. Заголовок ошибки

2. Описание ошибки

3. Начальные условия

4. Шаги воспроизведения

5. Ожидаемый результат

6. Фактический результат

7. Вложения

Наиболее эффективным описанием считается описание, которое отвечает на три вопроса:

— Что произошло?

— Где появилась ошибка?

— Когда или при каких условиях найден дефект?  
Пример плохого заголовка: Ошибка, когда нажимаю «Отправить сообщение».

Пример хорошего заголовка: При нажатии на кнопку «Отправить сообщение» в форме обратной связи сообщение не отправляется.  
 **Описание ошибки (Summary)**

Попробуйте в паре предложений описать суть дефекта, а также когда он появляется и в чем выражен.

Правильное и качественное описание также позволяет сразу понять проблему и приступить к ее исправлению.

Пример плохого описания: Жму «Отправить сообщение», а в ответ тишина.

Пример хорошего описания: При нажатии на кнопку "Отправить сообщение" в заполненной форме обратной связи ничего не происходит. Аналогичное поведение, если форма не заполнена.

**Начальные условия (Precondition)**

В случае, если есть специфичные действия или шаги воспроизведения достаточно объемные, то указываются начальные условия. Например:

1. Быть авторизованным в системе.

2. Находиться на главной странице.

Пример плохих начальных условий: Находиться на сайте.

Пример хороших начальных условий:

1. Страница "Контакты",

2. Платформа и устройство не имеют значения.

**Шаги воспроизведения (Steps To Reproduce)**

Шаги, при которых повторяется найденная ошибка. Например:

1. Нажать на кнопку “Войти”

2. Ввести “Имя пользователя” и “Пароль”

3. Нажать на кнопку “Ок”

Убедитесь, что у вас нет лишних или ненужных шагов воспроизведения, которые будут отвлекать и тратить время команды.

Пример плохих шагов: 1. Зайти на сайт, 2. Зайти на страницу обратной связи, 3. Поставить курсор в поле Имя, 4. Ввести имя, 5. Поставить курсор в поле e-mail, 6. Ввести действующий e-mail, 7. Навести курсор на Отправить сообщение, 8. Щелкнуть по Отправить сообщение.

Пример хороших шагов:

1. Заполнить поля формы обратной связи,

2. Нажать на копку "Отправить сообщение»

**Ожидаемый результат (Expected Result)**

Результат, который должен быть при выполнении шагов. В идеале, его можно найти в ТЗ (техническом задании). На практике, ТЗ бывает не всегда и ожидаемый результат определяется либо здравым смыслом, либо по аналогии.

Пример плохого ожидаемого результата: Что-то должно произойти.

Пример хорошего ожидаемого результата: Сообщение отправляется либо система сообщает о невозможности его отправки.

**Фактический результат (Actual Result)**

Указывается результат, который получил тестировщик при выполнении описанных шагов. Также может отвечать на три вопроса “Что? Где? Когда?”.

Пример плохого фактического результата: Ничего нет.

Пример хорошего фактического результата: Сообщение не отправляется, не появляется ошибка об отправке. После нажатия на кнопку ничего не происходит.

**Вложения (Attachments)**

При необходимости тестировщики прикладывают скриншоты или видео воспроизведения ошибки. Также могут прикладывать логи выполнения программы.  
Атрибуты баг-репорта

**Серьезность (Severity)** — это атрибут, характеризующий влияние дефекта на работоспособность приложения. Проставляется специалистом по тестированию.

Серьезность имеет несколько параметров в зависимости от типа дефекта. Ее степень зависит от того, как она влияет на бизнес-логику (реализацию правил программы).

S1 – Блокирующая (Blocker). Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате которого дальнейшая работа с тестируемой системой или ее функциями становится невозможна.

S2 – Критическая (Critical). Критическая ошибка, неправильно работающая бизнес-логика, проблема, приводящая в нерабочее состояние некоторую часть системы, но есть возможность для работы с тестируемой функцией, используя другие входные точки.

S3 – Значительная (Major). Значительная ошибка, часть 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 по средствам пользовательского интерфейса, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.  
Серьезность и приоритет багов

**Серьезность** — это показатель влияния бага на работу программы, того, может ли она функционировать без исправления или баг ломает всю систему. Выделяют пять уровней серьезности багов:

S0 Trivial (Тривиальный) — баг не влияет на работу программы, поэтому для его исправления могут не выделить отдельную задачу, а исправить попутно при исправлении других, похожих ошибок. Например, при заполнении анкеты в поле «Дата рождения» по умолчанию отображается не актуальный год, а 1999-й.

S1 Minor (Незначительный) — баг почти не нарушает логику процессов, поэтому с ним программа может нормально работать. Например, неудобная навигация в интерфейсе.

S2 Major (Серьезный) — баг создает неудобства в использовании, но еще не нарушает функционал программы.

S3 Critical (Критический) — баг мешает приложению выполнять основные функции: калькулятор расходов неправильно считает бюджет или в текстовом редакторе невозможно вводить текст.

S4 Blocker (Блокирующий) — ситуация, когда программа не работает в принципе: сайт выдает «ошибку 404» или не запускается приложение.

Приоритет — это срочность выполнения задачи. Всего выделяется три уровня приоритетов:

P1 Высокий — исправляется в первую очередь, так как баг ломает работу приложения.

P2 Средний — обязательный к исправлению баг после критического.

P3 Низкий — не требует немедленного решения.  
**Жизненный цикл бага**

Статус бага в репорте определяется его «жизненным циклом», который состоит из четырех основных стадий:

Открыт (Open) — тестировщик выявил баг и добавил в репорт.

В работе (In Progress) — о баге сообщили исполнителю, и он занимается исправлением.

Исправлен (Ready for check) — исполнитель закончил работу по исправлению бага и передал проект на повторную проверку тестировщику.

Закрыт (Closed) — баг устранен и больше не воспроизводится.

Кроме основных есть еще несколько статусов:

Отклонен (Rejected) — исправлению бага помешала ошибка в репорте, например неверный алгоритм в пункте «Шаги к воспроизведению».

Отсрочен (Deferred) — баг признан неприоритетным и исправление переносится.

Переоткрыт (Reopened) — баг был отсрочен или отклонен, но теперь исполнитель взял его в работу.  
На что стоит обратить внимание при **описании дефекта**?

Выявить причину возникновения. Например, если на сайте не получается восстановить пароль, то проблема может быть как в бэкенде, так и во фронтенде. Задача тестировщика — разобраться в ней, так как от этого зависит, кому из разработчиков отдавать баг на исправление.

Провести проверку на разных устройствах. Если проблема есть в десктоп-версии, то она может возникнуть и на мобильных устройствах, поэтому стоит проверить.

Провести проверку в разных версиях ПО. Баг может не воспроизводиться в старой версии ПО, но появится в новой.

Описать несоответствие ожидаемому результату. Чтобы сопоставить то, как работает программа сейчас, с ожидаемым результатом, начинающим специалистам лучше свериться с технической документацией и техническим заданием, где подробно описано, как все работает в идеале  
  
  
Несколько примеров проблем и попробуем правильно определить их Серьезность. Чтобы было понятно, представим, что мы тестируем приложение по заказу такси.  
Приложение «падает» при попытке найти свободное такси.

Чтобы правильно поставить Серьезность, необходимо определить влияние ошибки на дальнейшую работу функционала. Из названия видно, что после появления ошибки приложение перестает работать. Значит, влияние высокое.

Сразу же отбрасываем Тривиальную и Незначительную Серьезность, так как из их описания понятно, что ошибка не должна сильно влиять на приложение.

У нас остается только три варианта: Значительная, Критическая и Блокирующая серьезности.

Подходит ли нам Значительная Серьезность? Очевидно, что нет. Во-первых, ошибка достаточно критична. Во-вторых, другим способом найти такси мы не можем, т.е. нет возможности работы с тестируемой функцией, используя другие входные точки. Более того, функционал работает не некорректно, а не работает вообще.

Остаются Критическая и Блокирующая серьезности. В нашем случае Блокирующая подходит больше, так как часть функционала не работает и нет других возможностей найти такси. Следовательно, мы выставляем Блокирующую Серьезность.

Невозможно указать адрес назначения с помощью “Указать на карте”.

Снова начинаем рассуждать. Тривиальная и Незначительная не подходят, потому что ошибка в какой-то мере нарушают бизнес логику работы приложения. Блокирующую можно не брать, т.к. функционал в целом работает и его можно использовать через другую точку входа, а именно ввести адрес вручную. Остается только два варианта: Критическая и Значительная. Мы уже сказали, что проблема не приводит к полной неработоспособности части функционала. Тем не менее это значительная ошибка, т.к. функционал частично не работает, следовательно остается только вариант Значительная. Его мы и укажем.  
Приоритет

Приоритет отличается от Серьезности тем, что указывает когда необходимо исправить ошибку.

**Приоритет (Priority)** – это атрибут, указывающий на очередность выполнения задачи или устранения дефекта. Проставляется руководителем или менеджером проекта.

P1 – Высокий (High) – требуется исправить в первую очередь.

P2 – Средний (Medium) – требуется исправить во вторую очередь, когда нет дефектов с высоким приоритетом.

P3 – Низкий (Low) – исправляется в последнюю очередь, когда все дефекты с более высоким приоритетом уже исправлены.

С помощью приоритета менеджер проекта говорит, когда стоит исправить найденную проблему.