

HW QA Theory

Виды тестирования (ДЗ Вадим Ксендзов)

Stanislav

08.12.2021

**Вопросы, которые нужно оформить:**

* классификация тестирования по уровню детализации
* по доступу к коду и архитектуре приложения (методы)
* по запуску кода на исполнение (статическое, динамическое)
* по степени важности тестовых функций (виды тестирования, связанные с изменениями: smoke, regression, end2end, retest, new feature testing, критический путь)
* по степени автоматизации (ручное, автоматизация)
* по принципу работы с приложениями (позитивное, негативное)

## Классификация тестирования по уровню детализации приложения (по уровню тестирования)

* Модульное тестирование,
* Интеграционное тестирование,
* Системное тестирование.

**Модульное (компонентное) тестирование** - (unit testing, module testing) направлено на проверку отдельных небольших частей приложения, которые как правило, можно исследовать изолированно от других подобных частей. При выполнении данного тестирования могут проверяться отдельные функции или методы классов, сами классы, взаимодействие классов, небольшие библиотеки, отдельные части приложения. Часто данный вид тестирования делают с помощью средств автоматизации, которые значительно ускоряют разработку соответствующих тест-кейсов.

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*](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок. Например, обновить используемую в проекте библиотеку до актуальной версии можно в любой момент, прогнав тесты и выявив несовместимости.

Его цель заключается в том, чтобы проверить, что каждая единица программного кода работает должным образом. Данный вид тестирование выполняется разработчиками на этапе кодирования приложения. Модульные тесты изолируют часть кода и проверяют его работоспособность. Единицей для измерения может служить отдельная функция, метод, процедура, модуль или объект.

В моделях разработки SDLC, STLC, V Model модульное тестирование – это первый уровень тестирования, выполняемый перед интеграционным тестированием. Модульное тестирование – это метод тестирования WhiteBox, который обычно выполняется разработчиком. На деле же из-за нехватки времени или халатности разработчиков, иногда модульное тестирование приходится проводить QA инженерам.

**Интеграционное тестирование** (integration testing) - направлено на проверку взаимодействия между несколькими частями приложения (каждая из которых, в свою очередь, проверена отдельно на стадии модульного тестирования). К сожалению, даже если мы работаем с очень качественными отдельными компонентами, “на стыке” их взаимодействия часто возникают проблемы. Именно эти проблемы и выявляет интеграционное тестирование.

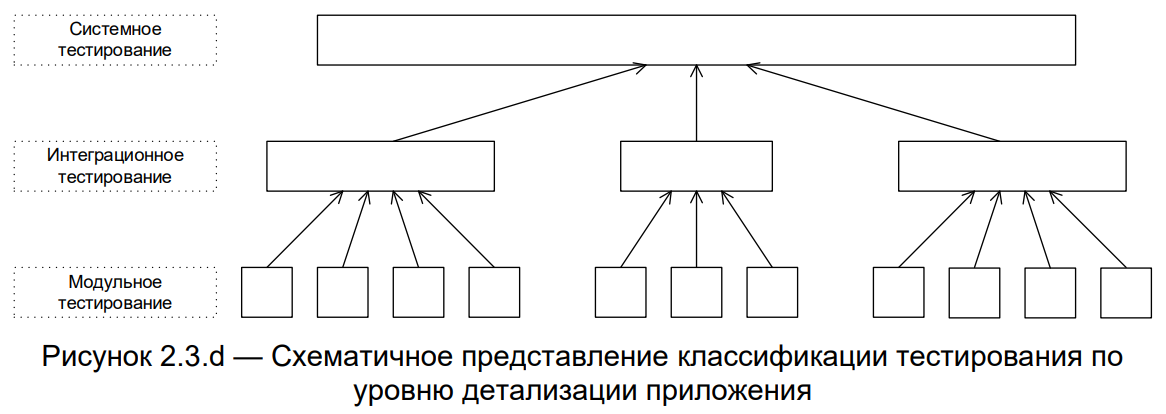
Интеграционное тестирование – это тип тестирования, при котором программные модули объединяются логически и тестируются как группа. Как правило, программный продукт состоит из нескольких программных модулей, написанных разными программистами. Целью нашего тестирования является выявление багов при взаимодействии между этими программными модулями и в первую очередь направлен на проверку обмена данными между этими самими модулями. Именно поэтому оно также называется «I & T» (интеграция и тестирование), «тестирование строк» и иногда «тестирование потоков».

**Системное тестирование** (system testing) - направлено на проверку всего приложения как единого целого, собранного из частей, проверенных на двух предыдущих стадиях. Здесь не только выявляются дефекты “на стыках” компонентов, но и появляется возможность полноценно взаимодействовать с приложением с точки зрения конечного пользователя, применяя множество других видов тестирования, перечисленных в данной главе.

**Системное тестирование** – это тестирование программного обеспечения выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям, как функциональным, так и не функциональным.

**Систе́мное тести́рование програ́ммного обеспе́чения**[[1]](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#cite_note-1) — это [тестирование программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) (ПО), выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам [тестирования чёрного ящика](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D1%87%D1%91%D1%80%D0%BD%D0%BE%D0%B3%D0%BE_%D1%8F%D1%89%D0%B8%D0%BA%D0%B0), и, тем самым, не требует знаний о внутреннем устройстве системы.

[Альфа-тестирование](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D1%8C%D1%84%D0%B0-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [бета-тестирование](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D1%82%D0%B0-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) являются подкатегориями системного тестирования. (Потов в конце идет E2E, но почему то тут не пишут)



## Классификация по доступу к коду и архитектуре приложения

* Метод белого ящика,
* Метод черного ящика,
* Метод серого ящика.

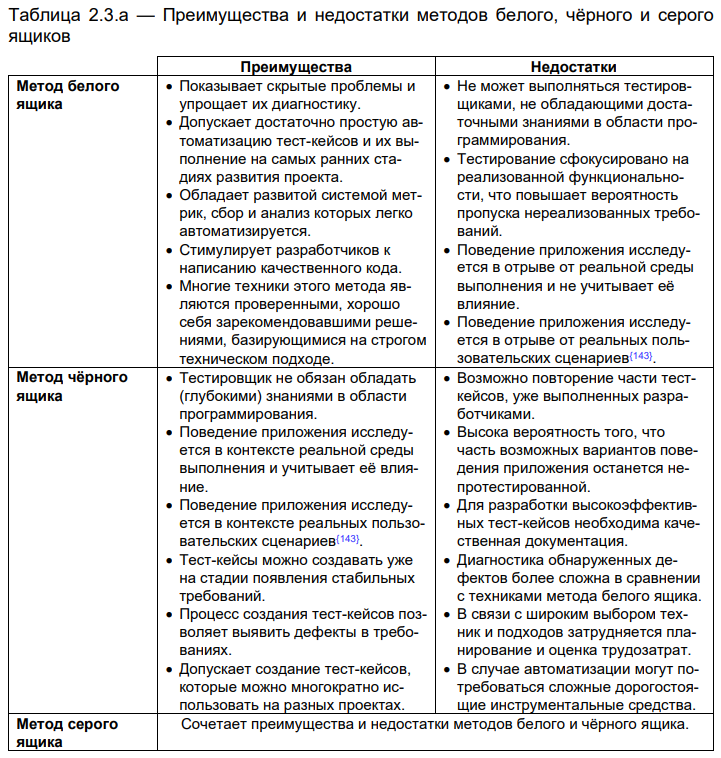
**Метод белого ящика** (white/glass box testing) -у тестировщика есть доступ к внутренней структуре и коду приложения, а так же есть достаточно знаний для понимания увиденного. В основном проводится на уровне модулей (юнит тестирование). Выполняют как правило разработчики, нужно знание кода.



**Метод черного ящика** (black box testing) -у тестировщика либо нет доступа к внутренней структуре и коду приложения, либо не достаточно знаний для их понимания, либо он сознательно не обращается к ним в процессе тестирования. При этом абсолютное большинство видов тестирования работают по методу черного ящика, идею которого можно сформулировать так: тестировщик оказывает на приложения воздействия (и проверяет реакцию) тем же способом, каким при реальной эксплуатации приложения на него воздействовали бы другие пользователи и приложения. Основной информацией для создания тест-кейсов выступает документация (особенно - требования и общий здравый смысл).

Система, которую представляют как «чёрный ящик», рассматривается как имеющая некий «вход» для ввода информации и «выход» для отображения результатов работы, при этом происходящие в ходе работы системы процессы наблюдателю неизвестны. Предполагается, что состояние выходов функционально зависит от состояния входов.

**Метод серого ящика** (gray box testing) -комбинация методов черного и белого ящиков, состоящая в том, что к части кода и архитектуры у тестировщика доступ есть, а к части - нет. Его явное упоминание крайне редкий случай: обычно говорят о методах белого или черного ящика в применении к тем или иным частям приложения, при этом понимая, что “приложение целиком” тестируется по методу серого ящика. Методы белого и черного ящика не являются конкурирующими или взаимоисключающими - напротив, они гармонично дополняют друг друга, компенсируя таким образом имеющиеся недостатки.



## Классификация по запуску кода на исполнение

* Статическое тестирование,
* Динамическое тестирование

**Статическое тестирование** (static testing) - тестирование без запуска кода на исполнение. В рамках этого подхода тестированию могут подвергаться:

* Документы (требования, тест-кейсы, описания архитектуры приложения, схемы баз данных и т.д.)
* Графические прототипы (например, эскизы пользовательского интерфейса)
* Код приложения (что часто выполняется самим программистами в рамках аудита кода (code review))
* Параметры (настройки) среды исполнения приложения
* Подготовленные тестовые данные

Статическое тестирование начинается на ранних этапах жизненного цикла ПО и является, соответственно, частью процесса верификации. Для этого типа тестирования в некоторых случаях даже не нужен компьютер – например, при проверке требований.

Большинство статических техник могут быть использованы для «тестирования» любых форм документации, включая вычитку кода, инспекцию проектной документации, функциональной спецификации и требований.

**Динамическое тестирование** (dynamic testing) - тестирование c запуском кода на исполнение. Запускаться на исполнение может как код всего приложения целиком (системное тестирование), так и код нескольких взаимосвязанных частей (интеграционное тестирование), отдельных частей (модульное тестирование) и даже отдельные участки кода. Основная идея этого вида тестирования состоит в том, что проверяется реальное поведение (части) приложения.

Таким образом, анализируется *поведение* программы во время ее работы. Для выполнения динамического тестирования необходимо чтобы тестируемый программный код был *написан*,*скомпилирован* и *запущен*. При этом, может выполняться проверка внешних параметров работы программы: загрузка процессора, использование памяти, время отклика и т.д. – то есть, ее производительность.

Динамическое тестирование является частью процесса валидации программного обеспечения.

## Классификация по (убыванию) степени важности тестируемых функций (по уровню функционального тестирования) / “по глубине тестирования”

* Дымовое тестирование,
* Тестирование критического пути,
* Расширенное тестирование.

**Дымовое тестирование** (smoke test) - направлено на проверку самой главной, самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования приложения (или иного объекта, подвергаемого дымовому тестированию).

Дымовое тестирование проводится после выхода нового билда, чтобы определить общий уровень качества приложения и принять решение о (не)целесообразности выполнения тестирования критического пути и расширенного тестирования. Поскольку тест-кейсов на уровне дымового тестирования относительно немного, а сами они достаточно просты, но при этом очень часто повторяются, они являются хорошими кандидатами на автоматизацию. В глоссарии ISTQB сказано: «sanity test: See smoke test». Пороговое значение метрики прохождения этого теста: 100%

*Дымовое тестирование рассматривается как короткий цикл тестов, выполняемый для подтверждения того, что после сборки кода (нового или исправленного) устанавливаемое приложение, стартует и выполняет основные функции*.

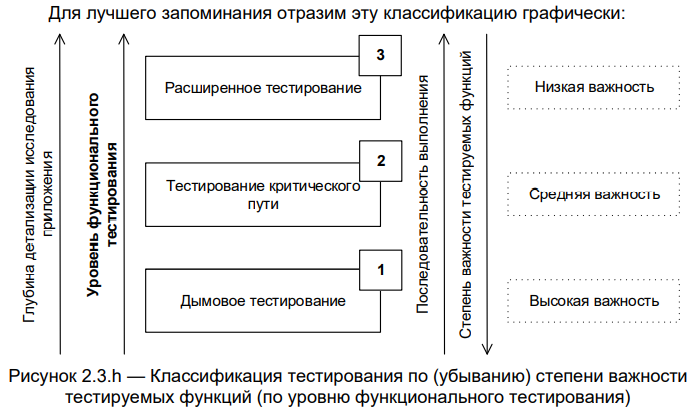
В некоторых источниках ошибочно полагают, что санитарное и [**дымовое тестирование**](http://www.protesting.ru/testing/types/smoke.html) - это одно и тоже. Мы же полагаем, что эти виды тестирования имеют "вектора движения", направления в разные стороны. В отличии от дымового (*Smoke testing*), **санитарное тестирование** (*Sanity testing*) **направлено вглубь** проверяемой функции, в то время как **дымовое направлено вширь**, для покрытия тестами как можно большего функционала в кратчайшие сроки.

**Тестирование критического пути** (Critical path test) - направлено на исследование функциональности, используемой типичными пользователями в типичной повседневной деятельности. Пороговое значение метрики прохождения этого теста уже ниже: 70-80-90%

Чаще всего на практике, на данном уровне тестирования проверяется основная масса требований к продукту. Пример: выбор шрифта, возможность набора текста, вставки картинок и т.д. Позитивный тест критического пути — это проверка работоспособности функций программного продукта, с которыми пользователь сталкивается ежедневно. Негативный тест критического пути — это проверка всевозможных вариантов нестандартного использования функциональности, используемой пользователем каждый день. Один из самых распространенных видов функционального тестирования. Частота зависит от необходимости проверки всего приложения в сжатые сроки.

**Расширенное тестирование** (Extended test) - направлено на исследование всей заявленной в требованиях функциональности - даже той, которая низко проранжирована по степени важности. При этом учитывается какая функциональность более, а какая менее важная. Но при достаточном количестве времени и иных ресурсов тест-кейсы этого уровня могут затронуть даже самые низкоприоритетные требования.

Еще одним направлением расширенного тестирования являются нетипичные, маловероятные, экзотические случаи и сценарии использования функций и свойств приложения, затронутых на предыдущих уровнях. Пороговое значение метрики прохождения этого теста уже ниже: 30-50%, т.к. подавляющее большинство найденных здесь дефектов не представляют угрозы для успешного использования приложения большинством пользователей. Как позитивные так и негативные тесты могут встречаться на всех перечисленных уровнях.



## Классификация по целям и задачам (продолжение)

* Регрессионное тестирование,
* Повторное тестирование (re-testing)
* Сквозное тестирование (e2e, end-to-end) - нет у С.Куликова
* Тестирование новой функциональности (new feature testing)

**Регрессионное тестирование** (regression testing) - тестирование направленное на проверку того факта, что в ранее работоспособной функциональности не появились ошибки, вызванные изменениями в приложении или в среде его функционирования. “Фундаментальная проблема при сопровождении программ состоит в том, что исправление одной ошибки с большой вероятностью (20-50%) влечёт появление новой”. Поэтому регрессионное тестирование является неотъемлемым инструментом обеспечения качества и активно используется практически в любом проекте.

**Регрессио́нное тести́рование** **-**  собирательное название для всех видов [тестирования программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), направленных на обнаружение ошибок в уже протестированных участках [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4). Такие ошибки — когда после внесения изменений в программу перестаёт работать то, что должно было продолжать работать, — называют *регрессионными ошибками* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *regression bugs*).

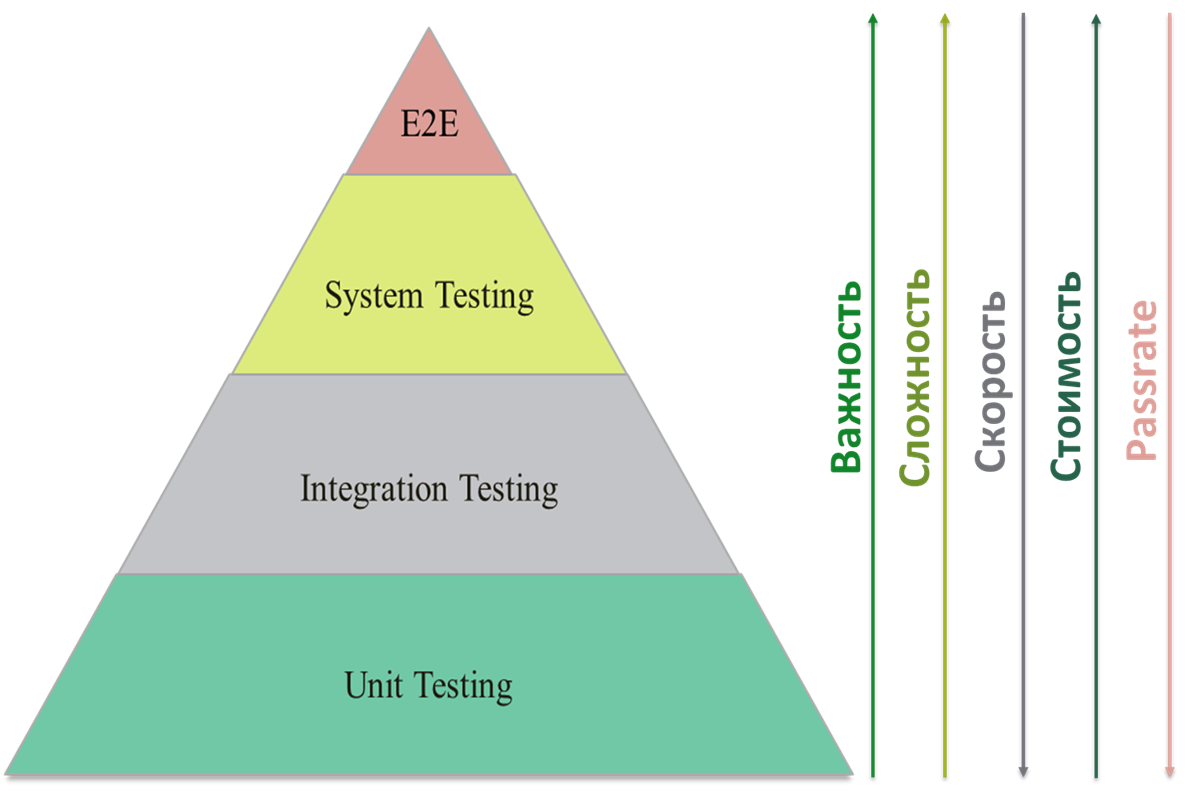
**Повторное тестирование** (re-testing) - выполнение тест-кейсов, которые ранее обнаружили дефекты, с целью подтверждения устранения дефектов. Фактически этот вид тестирования сводится к действиям на финальной стадии жизненного цикла отчета о дефекте, направленным на то, чтобы перевести дефект в состояние “проверен” и “закрыт”.

**Сквозное тестирование** (e2e, end-to-end) - это процесс тестирования, при котором происходит подробная эмуляция пользовательской среды. То есть при данном тестировании имитируют:

* щелчки мышью,
* нажатия на кнопки,
* заполнение форм,
* переходы по страницам и ссылкам,
* и другие поведенческие факторы.

Суть этого тестирования — посмотреть, так ли работает программа для конечного клиента, как рассчитывалось изначально? При этом нужно учитывать, что пользователю все равно, функционирует ли программа «как надо», ему главное, чтобы программа функционировала и оправдывала ожидания, поэтому основной упор делается на корректное функционирование.

Е2Е-процесс — это конечный этап тестирования, после него никакого тестирования не проводят. Он самый трудозатратный и дорогой, именно поэтому находится на вершине пирамиды тестирования.



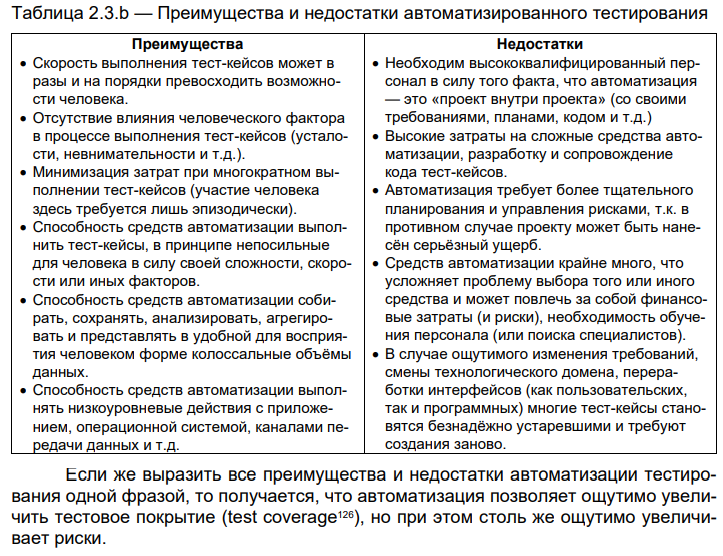
**Тестирование новой функциональности** (new feature testing) - в данном виде тестирования акцент делается на тестировании новой функциональности, появившейся в конкретном выпуске (build) программного продукта. New Feature Test (NFT, AT of NF) – определение качества поставленной на тестирование новой функциональности, которая ранее не тестировалась. Данный тип тестирования включает в себя: проведение полного теста (АТ) непосредственно новой функциональности; тестирование новой функциональности на соответствие документации; проверку всевозможных взаимодействий ранее реализованной функциональности с новыми модулями и функциями.

## Классификация по степени автоматизации

* Ручное тестирование,
* Автоматизированное тестирование

**Ручное тестирование** - (manual testing) - тестирование при котором тест-кейсы выполняются человеком вручную без использования средств автоматизации. Несмотря на то, что это звучит очень просто, от тестировщика зачастую требуются такие качества как терпеливость, наблюдательность, креативность, умение ставить нестандартные эксперименты, а также умение видеть и понимать, что происходит “внутри системы”, т.е. как внешние воздействия на приложение трансформируются в его внутренние процессы.

**Автоматизированное тестирование** (automated testing, test automation) - набор техник, подходов и инструментальных средств, позволяющий исключить человека из некоторых задач в процессе тестирования. Тест-кейсы частично или полностью выполняет специальное инструментальное средство, однако разработка тест-кейсов, подготовка данных, оценка результатов выполнения, написание отчетов об обнаруженных дефектах - всё это и много другое по прежнему делает человек.



## Классификация по принципу работы с приложением

* Позитивное тестирование,
* Негативное тестирование

**Позитивное тестирование** (positive testing) - направлено на исследование приложения в ситуации, когда все действия выполняются строго по инструкции без каких бы то ни было ошибок, отклонений, ввода неверных данных и т.д. Если позитивные тест кейсы завершаются ошибками, это тревожный признак - приложение работает не верно даже в идеальных условиях (а в неидеальных видимо еще хуже). Для ускорения тестирования несколько позитивных тест-кейсов можно объединять (например, перед отправкой заполнить все поля формы верными значениями) - иногда это может усложнить диагностику ошибки, но существенная экономия времени компенсирует этот риск.

*Позитивное тестирование* – это процесс проверки на корректное поведение согласно техническим требованиям и документации. Позитивное тестирование выполняется для обеспечения того, что система делает именно то, что ожидается. Например, ожидает от нас система положительного числа в поле цена, мы вводим 100 руб.

**Негативное тестирование** (negative testing) - направлено на исследование работы приложения в ситуациях, когда с ним выполняются (некорректные) операции и/или используются данные потенциально приводящие к ошибкам (классика жанра - деление на ноль). Поскольку в реальной жизни таких ситуаций значительно больше (пользователи допускают ошибки, злоумышленники осознанно “ломают” приложение, в среде работы приложения возникают проблемы и т.д.), негативных тест-кейсов оказывается значительно больше, чем позитивных (иногда в разы или даже на порядки). В отличии от позитивных, негативные тест-кейсы не стоит объединять, т.к. подобное решение может привести к неверной трактовке поведения приложения и пропуску (необнаружению) дефектов.

*Позитивные проверки* — это проверки с данными, введения которых продукт ожидает от пользователя. Например, ожидает от нас система положительного числа в поле цена, мы вводим 100 руб.

*Негативные проверки* — это, соответственно, те данные, которых программа не ждет. В примере с ценой в негативном тестировании мы введем в это поле буквы, символы и т.п.