МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВНИЯ

*"БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ"*

**Реферат**

**Основы тестирования программного обеспечения. Разновидности тестирования. Тестирование программы как "черного ящика” или функциональное тестирование.**

Выполнил:

студент БГУ

Факультета прикладной математики и информатики

1 курса

10 группы

Возовиков Никита Александрович

**Минск 2020**

***Содержание***

***1.Основы тестирования программного обеспечения***

***2. Разновидности тестирования***

***2.1.*** ***Виды тестирования по целям***

***2.2.*** ***По степени автоматизации***

***2.3.*** ***По позитивности сценария***

***2.4.*** ***По доступу к коду программного продукта***

***2.5.*** ***По уровню***

***2.6.*** ***По исполнителю***

***2.7.*** ***По формальности***

***2.8.По важности***

***3.Тестирование программы как "черного ящика” или функциональное тестирование***

***3.1.*** ***Тестирование программы как " черного ящика"***

***3.2.*** ***Функциональное тестирование***

***3.3.*** ***Цели функционального тестирования.***

***3.4.*** ***Уровни функционального тестирования***

***4.Эквивалентное разбиение***

***5. Анализ граничных значений***

***6. Применение функциональных диаграмм***

***7. Заключение***

***8. Источники***

## Основы тестирования программного обеспечения

**Тестирование** (*Testing*) – процесс, содержащий в себе все активности жизненного цикла, как динамические, так и статические, касающиеся планирования, подготовки и оценки программного продукта и связанных с этим результатов работ с целью определить, что они соответствуют описанным требованиям, показать, что они соответствуют описанным требованиям, показать, что они подходят для заявленных целей и для определения дефектов. Процесс, направленный на выявление существующих в ПО дефектов. Под дефектом здесь понимается участок программного кода, выполнение которого при определенных условиях приводит к неожиданному поведению системы (т.е. поведению, не соответствующему требованиям). Неожиданное поведение системы может приводить к сбоям в ее работе и отказам, в этом случае говорят о существенных дефектах программного кода. Некоторые дефекты вызывают незначительные проблемы, не нарушающие процесс функционирования системы, но несколько затрудняющие работу с ней. В этом случае говорят о средних или малозначительных дефектах. По статистике, мероприятия по тестированию занимают порядка 40% от всего процесса разработки продукта.

**Задача тестирования** – определение условий, при которых проявляются дефекты системы и протоколирование этих условий. В задачи тестирования обычно не входит выявление конкретных дефектных участков программного кода и никогда не входит исправление дефектов – это задача отладки, которая выполняется по результатам тестирования системы.

**Цель тестирования программного кода** – минимизация количества дефектов, в особенности существенных, в конечном продукте.

Тестирование само по себе не может гарантировать полного отсутствия дефектов в программном коде системы – тестирование программ может использоваться для демонстрации наличия ошибок, но оно никогда не покажет их отсутствие. Однако, в сочетании с процессами верификации и валидации, направленными на устранение противоречивости и неполноты проектной документации (в частности – требований на систему), грамотно организованное тестирование дает гарантию того, что система удовлетворяет требованиям и ведет себя в соответствии с ними во всех предусмотренных ситуациях.

**Верификация** (*verification*) – это процесс определения, выполняют ли программные средства и их компоненты требования, наложенные на них в последовательных этапах жизненного цикла разрабатываемой программной системы. Основная цель верификации состоит в подтверждении того, что программное обеспечение соответствует требованиям. Дополнительной целью является выявление и регистрация дефектов и ошибок, которые внесены во время разработки или модификации программы. Верификация является неотъемлемой частью работ при коллективной разработке программных систем. При этом в задачи верификации включается контроль результатов одних разработчиков при передаче их в качестве исходных данных другим разработчикам. Для повышения эффективности использования человеческих ресурсов при разработке, верификация должна быть тесно интегрирована с процессами проектирования, разработки и сопровождения программной системы.

**Валидация** (*validation*) программной системы – процесс, целью которого является доказательство того, что в результате разработки системы мы достигли тех целей, которые планировали достичь благодаря ее использованию. Иными словами, валидация – это проверка соответствия системы ожиданиям заказчика.

**Отладка** (*debugging*) – процесс, направленный на локализацию и устранение ошибок в системе.

Отличия понятий верификации и отладки: оба этих процесса направлены на уменьшение ошибок в конечном программном продукте, однако отладка – процесс, направленный на локализацию и устранение ошибок в системе, а верификация – процесс, направленный на демонстрацию наличия ошибок и условий их возникновения. Верификация, в отличие от отладки – контролируемый и управляемый процесс. Верификация включает в себя анализ причин возникновения ошибок и последствий, которые вызовет их исправление, планирование процессов поиска ошибок и их исправления, оценку полученных результатов.

## Разновидности тестирования

***1. Виды тестирования по целям***

Согласно с тем, какие цели вы преследуете, тестируя то или иное приложение или программу, тестирование бывает:

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

*Функциональное тестирование* направлено на проверку того, какие функции программного продукта реализованы, и того, насколько верно они реализованы.

*Нефункциональное* – проверяет корректность работы нефункциональных требований. Этот вид тестирования скорее проверяет как программный продукт работает и включает в себя следующие виды:

* **Тестирование производительности** – проверяет как программный продукт работает под определенной нагрузкой. Включает в себя тестирование:

>**Нагрузочное** – определяет, как программа работает под ожидаемой нагрузкой.

**> Стрессовое** – определяет работоспособность программного продукта при максимальной нагрузке.

**> Тестирование стабильности** – проверяет, выдержит ли программный продукт длительную нагрузку.

**> Конфигурационное** – определяет, как программный продукт будет работать в условиях смены конфигурации (платформы, драйверов, компьютеров).

* **Тестирование пользовательского интерфейса** – определяет удобство пользовательского интерфейса (кнопки, цвета, выравнивание и т.д.).
* **Тестирование удобства использования** – проверяет, удобен ли программный продукт в использовании.
* **Тестирование защищенности** – определяет, насколько безопасно использование программного продукта, т.е. защищен ли программный продукт от атак хакеров, несанкционированного доступа к данным и т.д.
* **Инсталляционное тестирование** – проверяет, не возникает ли проблем при установке, удалении, а также обновлении программного продукта.
* **Тестирование совместимости** – тестирование работы программного продукта в определенном окружении.
* **Тестирование надежности** – проверяет работу приложения 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** – тестирование программного продукта без использования дополнительных программных средств, т.е. тестирование «вручную».
* **Автоматизированное** – тестирование программного продукта с использованием программных средств.

***3. По позитивности сценария***

По позитивности сценария тестирование бывает:

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

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

В процессе тестирования тестировщик может работать с программным продуктом, 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 программного продукта. В данном случае тестировщик не работает с кодом программного продукта, но он знаком с внутренней структурой программы или приложения и взаимодействием между компонентами.

***5. По уровню***

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

***6. По исполнителю***

* **Альфа-тестирование** – тестирование программного продукта на поздней стадии разработки. Проводится разработчиками или тестировщиками.
* **Бета-тестирование** – тестирование программного продукта 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:

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

## Тестирование программы как "черного ящика” или функциональное тестирование

***Тестирование программы как " черного ящика"*** - это тип тестирования, в котором функциональные возможности программного обеспечения тестируются без каких-либо ссылок на внутренний дизайн, код или алгоритм, используемый в программе.

Тестирование по стратегии чёрного ящика заключается в выборе соответствующих данных в соответствии с функциональными возможностями системы и ее тестирования в отношении функциональных спецификаций для того, чтобы проверить наличие нормального и ненормального поведения системы. В наше время многие компании для получения точных результатов передают испытание своих работ третьим лицам. Это происходит потому, что Разработчик системы очень хорошо понимает внутреннюю логику и кодирование системы, что делает его непригодным для тестирования его приложения. В целях реализации стратегии тестирования черного ящика, тестировщик должен иметь полное представление о спецификации требований системы, и ее поведения в ответ на конкретные действия.

Итак, говоря о тестировании методом черного ящика, мы говорим о функциональном тестировании (functional testing). Функциональное тестирование еще называют поведенческим или тестированием на поведенческом уровне. Тестировщики, занимающиеся функциональным тестированием, используют преимущественно метод черного ящика, а программисты, перед тем как передать приложение в тестирование, проверяют свои модули методом белого ящика.

***Функциональное тестирование*** — один из процессов жизненного цикла программного продукта, который проводится с целью получения объективных доказательств функционирования программного продукта в соответствии с установленными либо подразумеваемыми заказчиком требованиями к программному продукту. В процессе функционального (а также любого другого) тестирования тестировщик ищет дефекты (defect or bug).

Функциональное тестирование подразделяется на *ручное (manual testing)* и *автоматическое или автоматизированное (automated testing).*

Ручное тестирование подразумевает выполнение тестов вручную. В свою очередь автоматическое тестирование подразумевает привлечения каких-либо средств или инструментов для автоматизирования тестирования.

***Цели функционального тестирования.***

Тестирование осуществляется с тем, чтобы:

– обнаружить ошибки и задокументировать их;

– определить, соответствует ли приложение предъявляемым к нему требованиям.

– принять объективное заключение о возможности поставки программного продукта заказчику, и документирование этого заключения.

Тестировщики не принимают окончательного решения о готовности программного продукта. Как правило, это делает менеджер проекта, или решает сам заказчик. Однако тестировщик может повлиять на принятие решения, предоставляя полную и максимально объективную информацию о том, в каком состоянии находится продукт и каков уровень его качества на данный момент.

***Уровни функционального тестирования***

***Приемочный тест*** — это самый первый и короткий тест, проверяющий работу основной функциональности программного продукта. Данный тест длится от получаса до 2-3-х часов максимум в зависимости сложности программы, по результатам которого ведущий инженер по тестированию принимает решение о целесообразности дальнейшего тестирования. Если программа не прошла приемочный тест, она отправляется на доработку к программистам.

***Критический тест*** — основной вид теста, во время которого проверяются основная функциональность программного продукта критичная для конечного пользователя, при стандартном его использовании. В рамках данного тестирования, как правило, проверяется большинство требований предъявляемых к программному продукту.

***Расширенный тест*** — это углубленный тест, при котором проверяется нестандартное использование программного продукта. Прогоняются различные сложные, логически запутанные сценарии, совершаются действия, которые конечный пользователь будет совершать очень редко. Тестирование этого уровня требуется далеко не для всех типов приложений и во многих случаях могут не проводиться или проводиться ограниченно.

## Эквивалентное разбиение

Метод эквивалентного разбиения заключается в следующем. Область всех возможных наборов входных данных программы по каждому параметру разбивают на конечное число групп - ***классов эквивалентности.*** Наборы данных такого класса объединяют по принципу обнаружения одних и тех же ошибок: если набор какого-либо класса обнаруживает некоторую ошибку, то предполагается, что все другие тесты этого класса эквивалентности тоже обнаружат эту ошибку и наоборот. Разработку тестов методом эквивалентного разбиения осуществляют в два этапа:

**>на первом** -выделяют классы эквивалентности

**>на втором** - формируют тесты.

Выделение классов эквивалентности является эвристическим процессом, однако целесообразным считают выделять в отдельные классы эквивалентности наборы, содержащие допустимые и недопустимые значения некоторого параметра. При этом существует ряд правил:

• если некоторый параметр х может принимать значения в интервале

[1, 999], то выделяют

один правильный класс 1 ≤ х ≤ 999 и два неправильных: х < 1 и х>999;

• если входное условие определяет диапазон значений порядкового типа, например, «в автомобиле могут ехать от одного до шести человек», то определяется один правильный класс эквивалентности и два неправильных: ни одного и более шести человек;

• если входное условие описывает множество входных значений и есть основания полагать, что каждое значение программист трактует особо, например, «типы графических файлов: bmp, jpeg, vsd», то определяют правильный класс эквивалентности для каждого значения и один неправильный класс, например, txt;

• если входное условие описывает ситуацию «должно быть», например, «первым символом идентификатора должна быть буква», то определяется один правильный класс эквивалентности (первый символ -буква) и один неправильный (первый символ - не буква);

• если есть основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс разбивается на меньшие классы эквивалентности.

## Анализ граничных значений

Опыт показывает, что тесты, исследующие граничные условия, приносят большую пользу чем тесты, которые их не исследуют.

Граничные условия - это ситуации, возникающие непосредственно на, выше или ниже границ входных и выходных классов эквивалентности.

Анализ граничных значений отличается от эквивалентного разбиения в двух отношениях:

- выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных значений осуществляется таким образом, чтобы проверить тестом каждую границу этого класса.

- при разработке тестов рассматривают не только входные условия (пространство входов), но и пространство результатов, то есть выходные классы эквивалентности.

Метод анализа граничных значений требует творческого подхода и специализации. Приведем несколько общих правил этого метода :

1. Построить тесты для границ области и тесты с неправильными входными данными для ситуаций незначительного выхода за границы области, если входное условие описывает область значений.

Пример: если правильная область входных значений есть [-1.0 ; 1.0], то следует написать тесты для ситуаций: -1.0; 1.0; -1.001; 1.001.

1. Если входные условия удовлетворяют дискретному ряду значений, построить тесты для минимального и максимального значений условий и тесты для значений входных условий больших или меньших максимального и минимального.

Пример: если входной файл может содержать от 1 до 255 записей, то построить тесты для 0; 1; 255; 256 записей.

1. Использовать правило 1 для каждого входного условия.

Пример: Программа вычисляет ежемесячный расход. Минимум расхода составляет 0.00, а максимум - 1165.25 (усл. ед.). Нужно построить тесты для отрицательного расхода и расхода, большего 1165.25.

Важно проверить границы пространства результатов, поскольку не всегда границы входных областей представляют такой же набор условий, как и границы выходных областей. Пример: y=sin(x); x∈[0;π]

Не всегда можно получить результат вне выходной области, но, тем не менее, стоит рассмотреть эту возможность.

1. Построить тесты для минимального и максимального значения каждого выходного условия, а также тесты, обеспечивающие нарушение границ выходных условий.

Пример: система информационного поиска отображает на экране терминала наиболее релевантные рефераты в зависимости от входного запроса, но никак не более четырех рефератов. Тесты:

* 1. программа отображает ноль рефератов
  2. - один
  3. - четыре
  4. - пять (ошибочная ситуация)

5. Если вход или выход программы есть упорядоченное множество (последовательный файл, линейный список, таблицы), то сосредоточить внимание на первом и на последнем элементах одного множества.

Существенное различие между анализом граничных значений и эквивалентным разбиением состоит в том, что анализ граничных значений исследует ситуации, возникающие на и вблизи границ эквивалентных разбиений.

## Применение функциональных диаграмм

Функциональными - называют диаграммы, отражающие взаимосвязи функций разрабатываемого ПО. Каждый блок такой диаграммы (Activity) соответствует некоторой функции, для которой должны быть определены: исходные данные, результаты, управляющая информация и механизмы ее осуществления. Все связи на диаграмме представляются дугами, причем тип связи и ее направление строго регламентированы. Дуги, изображающие каждый тип связей, должны подходить к блоку с определенной стороны, а направление связи должно указываться стрелкой в конце.

Физически дуги исходных данных, результатов и управления представляют собой наборы данных, передаваемые между функциями. Дуги, определяющие механизм выполнения функции, в основном используются при описании спецификаций сложных информационных систем, которые включают как автоматизированные, так и ручные операции. Блоки и дуги маркируются текстами на естественном языке.

Блоки на функциональной диаграмме размещают по «ступенчатой» схеме в соответствии с последовательностью их работы или доминированием, которое понимается как влияние, оказываемое одним блоком на другие. В функциональных диаграммах SADT различают пять типов влияний блоков друг на друга:

>вход – выход блока подается на вход блока с меньшим доминированием, т.е. на вход следующего (рис. 3.1а);

> управление – выход блока используется как управление для блока с меньшим доминированием (следующего) (рис. 3.1б);

> обратная связь по входу – выход блока подается на вход блока с большим доминированием (на вход предыдущего) (рис. 3.1в);

·>обратная связь по управлению – выход блока используется как управляющая информация для блока с большим доминированием (предыдущего) (рис. 3.1г);

> выход – исполнитель – выход блока используется как механизм для другого блока (рис. 3.1д).

Дуги могут разветвляться и соединяться вместе различными способами. Разветвление означает, что часть или вся информация может использоваться в каждом ответвлении дуги.

Дуга всегда помечается до ветвления, чтобы идентифицировать передаваемый набор данных. Если ветвь дуги после ветвления не помечена, то непомеченная ветвь содержит весь набор данных. Каждая метка ветви уточняет, что именно содержит данная ветвь (рис. 3.2).

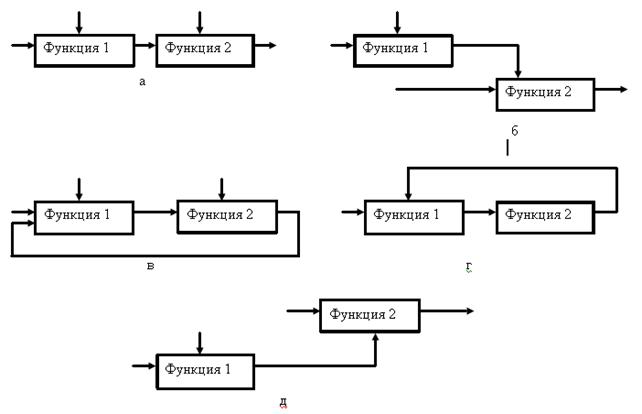


Рис. 3.1. Типы влияний блоков:

а – связь по входу; б – связь по управлению; в – обратная связь по входу;

г – обратная связь по управлению; д – связь выход-исполнитель.



Рис. 3.2. Пример обозначения дуг при ветвлении

Построение иерархии функциональных диаграмм ведется поэтапно с увеличением уровня детализации: диаграммы каждого следующего уровня уточняют структуру родительского блока. Построение модели предметной области (с точки зрения бизнес-процессов) начинают с блока самого верхнего уровня – контекстная диаграмма предметной области, для которого определяют исходные данные, результаты, управление и механизмы реализации. Затем этот блок последовательно детализируется с использованием метода пошаговой детализации. При этом рекомендуется каждую функцию представлять не более чем 3-7 блоками.

Во всех случаях каждая подфункция может использовать или продуцировать только те элементы данных, которые использованы или продуцируются родительской функцией, причем никакие элементы не могут быть опущены, что обеспечивает непротиворечивость построенной модели.

Стрелки, проходящие с родительской диаграммы или уходящие на нее, нумеруют, используя символы и числа. Символ, в этом случае, обозначает тип связи: I – входная, C – управляющая, М – механизм, R – результат. Число – номер связи по соответствующей стороне родительского блока, считая сверху вниз и слева направо.

Все диаграммы связаны друг с другом иерархической нумерацией блоков: первый уровень – А0 (А – от слов Activity), второй – А1, А2, и т.п., третий – А11, А12, А13 и т.п., где первые цифры – номер родительского блока, а последняя – номер конкретного субблока родительского блока.

Процесс детализации завершается после получения функций, назначение которых хорошо понятно как заказчику, так и разработчику. Для описания полученных функций используют естественный язык, либо псевдокод.

В процессе построения иерархии диаграмм фиксируют всю уточняющую информацию и строят словарь стрелок, в котором определяют структуры и элементы данных, показанных на диаграммах.

Словарь стрелок решает очень важную задачу. Диаграммы создаются аналитиком для того, чтобы провести сеанс экспертизы, т.е. обсудить содержимое диаграммы со специалистом предметной области. Как известно, в любой предметной области формируется профессиональный жаргон, причем очень часто жаргонные выражения имеют нечеткий смысл и поэтому воспринимаются разными специалистами по-разному. В то же время аналитик, как автор диаграмм, вынужден употреблять те выражения, которые наиболее понятны экспертам. Поскольку формальные определения часто сложны для восприятия, аналитик вынужден употреблять профессиональный жаргон, а чтобы не возникло неоднозначных трактовок, в словаре стрелок каждому понятию можно дать расширенное и, если это необходимо, формальное определение.

Таким образом, в результате функционального моделирования предметной области получают спецификацию, которая состоит из:

· иерархии функциональных диаграмм;

· спецификаций функций нижнего уровня;

· словаря стрелок (данных),

при этом, элементы полученной спецификации должны иметь ссылки друг на друга.

## Заключение

Как видим, человеческие ошибки могут приводить к появлению дефектов на всех стадиях разработки программного продукта, причем последствия этого могут быть самыми разными – от незначительных до катастрофических.

Тестирование позволяет находить и исправлять дефекты, тем самым снижая уровень риска и повышая качество продукта. Проверяются, в том числе, и места пользовательского интерфейса, где пользователь может сделать ошибку или неправильно понять вывод программы, а также устойчивость системы к злонамеренным действиям.

Почему важен процесс тестирования?

– процесс разработки ПО невозможен без контроля качества разрабатываемого продукта;

– процесс тестирования ПО представляет собой столь же неотъемлемую часть процесса разработки, как и проектирование;

– тестирование позволяет оценить качество разрабатываемого продукта.

## Источники

1. Тестирование программного обеспечения. Базовый курс. © EPAM Systems, 2015–2019. Святослав Куликов.
2. А. М. ДВОРЯНКИН, А. А. ЕРОФЕЕВ, А. В. АНИКИН ОСНОВНЫЕ МЕТОДЫ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
3. Котляров В.П. Основы тестирования программного обеспечения

# Тамре Луиза. Введение в тестирование программного обеспечения

1. С. Канер, Д. Фолк, Е. Нгуен. Тестирование программного обеспечения.
2. В.В. Липаев. Тестирование программ. - М.: Радио и связь, 1986.
3. <http://studepedia.org/index.php?vol=2&post=20752>
4. <https://studfile.net/preview/994030/page:6/>
5. <https://studopedia.ru/8_201409_metod-funktsionalnih-diagramm.html>
6. <https://quality-lab.ru/blog/extend-testing-of-boundary-values/>
7. <https://www.it-courses.by/boundary-value-analysis/>
8. <http://juice-health.ru/program/software-testing/496-black-box-testing>
9. <https://sites.google.com/site/testprogrammprodukt/funkcionalnoe-testirovanie>
10. <https://qa-academy.by/qaacademy/news/klassifikaciya-vidov-testirovaniya/>
11. <https://qaevolution.ru/testirovanie-po/vidy-testirovaniya-po/>
12. <https://habr.com/ru/post/279535/>