**ВВЕДЕНИЕ**

**§ 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

**1. РУЧНОЕ ТЕСТИРОВАНИЕ**

1.1. Виды ручного тестирования

1.2. Методы ручного тестирования

### 1.3. Основные инструменты ручного тестирования

1. **КЛАССИФИКАЦИЯ ВИДОВ ТЕСТИРОВАНИЯ**

2.1. Функциональное тестирование

2.2. Нефункциональное тестирование

**3. ПРИНЦИПЫ ТЕСТИРОВАНИЯ**

## 4. ТЕХНИКИ ТЕСТ-ДИЗАЙНА:

#### 4.1. Техники тестирования метода чёрного ящика

#### 4.2. Техники тестирования метода белого ящика

#### 4.3. техники тестирования, основанные на опыте

**5. МОДЕЛИ ПРОЦЕССА РАЗРАБОТКИ ПО**

5.1. Waterfall

5.2. Agile

5.2.1. Scrum

5.2.2. Kanban

# **6. АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ**

## 6.1. Алгоритм действий

## 6.2. Инструменты автоматизации тестирования

## § 2. ПРАКТИЧЕСКАЯ ЧАСТЬ

## 7. ИТОГИ ИССЛЕДОВАНИЯ

## 7.1. Анализ результатов тестирования

## 7.2. Реализация поставленных целей и задач

## 7.2. Рекомендации по улучшению качества ПО

## ПРИЛОЖЕНИЯ

## СПИСОК ЛИТЕРАТУРЫ

**ВВЕДЕНИЕ**

Белгосстрах – самая крупная в Республике Беларусь страховая компания. По всей Беларуси функционируют 8 филиалов и 108 представительств. Действующая сеть продаж страховых услуг Белгосстраха превышает более 5 000 точек продаж по всей территории страны, ряд из которых работает круглосуточно. Предоставляя широкий спектр услуг, страховая компания двигается в ногу со временем и развивает онлайн страхование. Для этого был создан онлайн личный кабинет, где пользователь может зарегистрироваться, и не только заключить договор страхования, но и управлять уже действующими страховками.

Данный дипломный проект посвящен анализу личного кабинета пользователя страховой компании Белгосстрах и разработке набора тестов для проверки соответствия модуля «форма регистрации» техническому заданию.

Дипломная работа посвящена подготовке конечного набора тестов, который должен покрывать поставленные требования, проверять модуль на корректность работы с точки заданного алгоритма функционирования и исключать появление программных ошибок, которые блокируют работу программы.

Основным практическим результатом выполнения дипломной работы является разработанная система ручных сценариев с частичной автоматизацией создания основных объектов тестируемого модуля. Данный набор тестов нашел практическое применение при последующем тестирование модуля «\_\_\_».

**Актуальность:** в настоящее Белгосстрах использует в своей работе программное обеспечение и ошибка в работе этих программ может принести большие неудобства, затраты этой страховой компании. Поэтому необходимо уделять много времени и ресурсов для тестирования этих программ.

**Цель тестирования**: проверка соответствия ПО страховой компании «Белгосстрах» предъявляемым требованиям, обеспечение уверенности в качестве ПО, поиск очевидных ошибок в программном обеспечения.

**Цель исследования:** спроектировать процесс тестирования программного обеспечения, в частности отдельного модуля «форма регистрации».

**Предмет исследования:** тестирование программного обеспечения, в частности отдельного модуля «форма регистрации».

**Задачи исследования:**

* найти и изучить материал по тестированию программного обеспечения;
* разработать тесты программного обеспечения;
* спроектировать процесс тестирования программного обеспечения.

Дипломный проект содержит \_\_\_листов из них:

\_\_\_рисунков,

\_\_таблиц,

\_\_\_приложение.

**§ 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

**1. РУЧНОЕ ТЕСТИРОВАНИЕ**

Ручное тестирование — это процесс поиска ошибок в программе без использования специальных ПО, силами человека. Тестировщик имитирует реальные действия пользователя и старается охватить максимум функций продукта и найти ошибки (на языке QA — «баги»). Специалист по QA ищет недоработки в визуален, функционале, логике ПО, проверяет его надежность и удобство. Все найденные ошибки QA фиксирует в баг-репорте — отчете о тестировании, по которому разработчики будут исправлять недочеты.

Любое приложение должно быть протестировано вручную прежде, чем автоматизировать процесс. Это необходимо для того, чтобы определить, целесообразно ли вообще внедрять автоматизацию. Для проведения ручного тестирования не нужно уметь пользоваться какими-либо инструментами. Один из фундаментальных принципов тестирования — **100% автоматизации невозможна**. Поэтому ручное тестирование неизбежно в каждом проекте.

Главная цель ручного тестирования — убедиться, что в приложении нет ошибок и что оно работает в полном соответствии с требованиями.

Для этого на стадии тестирования создаются тест кейсы, которые должны покрывать (в идеале) 100% функциональности тестируемого приложения.

Также ручной тестировщик проверяет, что обнаруженные баги исправляются разработчиками и повторно тестирует то, что было исправлено.

В целом, ручные тестировщики проверяют качество разрабатываемого приложения и обеспечивают доставку приложения максимально возможного качества конечным пользователям.

*Из каких шагов состоит ручное тестирование?*

1. **Читаем документацию и работаем с требованиями.**  На этом этапе QA-инженер может добавить требования, если они неполные, и сократить, если они невыполнимы.
2. **Планируем тестирование.** Определяем объем работы, бюджет, выбираем методы, типы и инструменты.
3. **Разрабатываем тестовые сценарии.** Специалисты создают тест-кейсы — алгоритм проверки ПО, а также чек-листы и готовят среду для выполнения тестов.
4. **Проводим первое тестирование.** Команда выполняет тесты и сообщает разработчикам об ошибках.
5. **Делаем повторное тестирование.**Когда программисты исправили ошибки, тестирование повторяют, чтобы проверить, что после изменений все работает.
6. **Готовим отчет о результатах.**В итоговом документе описывают все тесты, выполненные во время разработки программы.

**1.1. ВИДЫ РУЧНОГО ТЕСТИРОВАНИЯ**

**Модульное тестирование (Unit-тесты)** предполагает проверку отдельных компонентов ПО или частей кода. Это эффективный способ тестирования, если готовое приложение обновляют или дополняют функционалом. Если добавить новые модули, ошибки в них могут повлиять на работу других, уже налаженных и протестированных частей программы. Вместо того чтобы ломать сервис таким образом, можно сначала протестировать модуль отдельно, а потом добавить его в систему.

**Интеграционное тестирование (Integration Testing)**проверяет, как отдельные части приложения работают вместе. Часто бывает, что страницу авторизации и личный кабинет приложения программируют разные специалисты. Их инструменты и подходы могут отличаться, из-за этого конечный сервис может работать с ошибками. На этом этапе уже не нужно проверять отдельные элементы, например страницу авторизации, — вы уже сделали это unit-тестом. Здесь важно запустить разные элементы в группе и проверить, что они работают корректно. Например, что авторизация запускает процесс создания личного кабинета и все данные пользователя в нем отражаются правильно.

**Системное тестирование (System Testing)** нужно, чтобы понять, соответствует ли ПО исходным техническим требованиям. Это этап, когда модульные и интеграционные тесты уже прошли. Теперь время смотреть на готовый продукт, кликать по кнопкам, проверять, что все работает как задумано, сервисом удобно и приятно пользоваться.

**Приемочное тестирование (Acceptance Testing)** проверяет, подходит ли приложение под требования бизнеса. На этом этапе тестировщики исследуют поведение пользователей и производительность системы.

**1.2. МЕТОДЫ РУЧНОГО ТЕСТИРОВАНИЯ**

*Что такое черный, белый и серый ящики?*

Так называются методы тестирования. Они отражают то, сколько знает тестировщик о продукте на старте работы. Разберем каждый подход подробнее.

**Тестирование «черного ящика» (Black Box Testing) —** метод, в котором тестировщик ничего не знает о коде или структуре продукта. QA работает с программой как конечный пользователь. Этим методом проверяют функциональность: делает ли приложение то, что должно?

Например, в интернет- магазине важно проверить поиск товаров, фильтрацию результатов выдачи, возможность добавить продукты в корзину, ввести промокоды и оформить заказ. Иногда функции сервиса выглядят идеально в коде, но не работают на практике. В этом случае тестирование «черного ящика» помогает выявить баги, незаметные при проверке только кодовой части ПО.

**Тестирование «белого ящика» (White Box Testing),** также известное как glass box или прозрачное тестирование, — это, по сути, проверка исходного кода. Тестировщик анализирует блоки системы по отдельности и ищет проблемы.

Например, прозрачным тестированием можно проверить формы ввода контактов пользователя в интернет-магазине. Со стороны пользователя это выглядит так: вы нажали кнопку, email-адрес отправился в базу подписчиков магазина, вам на почту пришло письмо с промокодом на скидку. Если тестировать эту часть «черным ящиком», вы можете нажать на кнопку и не получить никакого письма. Зафиксировали баг, тест заканчивается. Методом «белого ящика» можно выявить, почему это происходит. QA-специалист смотрит, чтобы на уровне кода форма была надежно защищена от взлома и данные пользователей не утекли в руки мошенников. Также он следит, чтобы адрес почты отправился в базу данных, а дальше запустился процесс автоматической рассылки новостей об акциях и промокодах.

**Тестирование «серого ящика» (Grey Box Testing)** объединяет методы тестирования «белого» и «черного ящика». Цель этого подхода — найти любые ошибки в пользовательском интерфейсе или в разработке. У тестировщика нет доступа к коду приложения, но он знает общую структуру сервиса и его ограничения.

Для примера вернемся к форме в интернет-магазине. Например, при оформлении заказа нужно ввести имя и фамилию, тестировщику нужно проверить работу текстовых полей. QA знает, что у системы есть ограничение по длине фамилии, например, в 100 символов. Задача тестировщика — найти фамилии длиннее 100 символов (самая длинная в книге рекордов Гиннеса состоит из 700). Также он должен проверить, как будет вести себя система, если ввести в поле больше 100 букв. Приложение должно как минимум не ломаться и выдавать уведомление об ошибке.

### **1.3. ОСНОВНЫЕ ИНСТРУМЕНТЫ «РУЧНОГО» ТЕСТИРОВАНИЯ**

Тестирование – дорогой и трудоемкий этап разработки программных систем. Поэтому создан широкий спектр инструментальных средств для поддержки процесса тестирования, которые значительно сокращают расходы на него.

Ручное тестирование может проводиться в рамках интеграционного (взаимодействие с другими ПО и системами), регрессионного (тестирование различных изменений) и, конечно, при системном функциональном тестировании. Ручное тестирование помогает выявить и устранить «узкие места», снизить количество багов, обеспечить стабильность систем, оценить удобство эксплуатации, и, в конечном счете, получить продукт, удовлетворяющий ожидания пользователей.

Для ручного тестирования существуют специальные средства коллективной работы, позволяющие управлять командой и задачами. Вся коммуникация осуществляется прямо в программе:

* [JIRA](https://www.atlassian.com/software/jira)
* [Redmine](https://www.redmine.org/)
* [Bugzilla](https://www.bugzilla.org/)

Создание и анализ скриншотов с помощью специальных инструментов – важная часть работы инженера QA:

* [Snagit](https://www.techsmith.com/screen-capture.html)
* [Recordit API](http://help.recordit.co/api/)
* [Monosnap](https://monosnap.com/welcome)
* [GreenShot](https://getgreenshot.org/)

Для тестирования часто нужны различные наборы данных наборами данных. Для решения проблемы тестировщики применяют специальные инструменты-генераторы:

* [Mockaroo.com](https://www.mockaroo.com/)
* [Bugmagnet](https://bugmagnet.org/)

Тестирование юзабилити помогает выяснить, насколько продукт соответствует ожиданиям пользователей, и в то же время, экономят время и средства на проведение исследований:

* [User Zoom](https://www.userzoom.com/)
* [Reflector](https://reflector.en.softonic.com/)
* [Loop](https://www.loop11.com/)

В часто повторяющихся процессах со временем возникают пробелы. Чтобы ничего не упустить и не потерять, чек-лист – отличный инструмент. Тестировщик может им пользоваться для контроля, остальные участники процесса – для понимания происходящего:

* [Testpad](https://ontestpad.com/)
* [Sitechсo](https://sitechco.ru/)
* [Teamsuccess](https://teamsuccess.io/)

Мобильный эмулятор – это программный комплекс, позволяющий с точностью копировать функциональность мобильного устройства:

* [World Wide Web](http://validator.w3.org/)

Валидатор анализирует данные по URL приложения. Он проверяет правильность разметки веб-документов в HTML, совместимость с мобильными устройствами, работоспособность ссылок и многое другое:

* [BrowserStack](http://www.browserstack.com/)
* [CrossBrowserTesting](http://crossbrowsertesting.com/)
* [Mobile phone emulator](http://www.mobilephoneemulator.com/

В современном мире многие приложения очень часто используют API, для его тестирования используются следующие программы:

* [Postman](https://www.postman.com/)
* [SoapUI](https://www.soapui.org/)

Одним из важных этапов тестирования является тестирование ПО на безопасность. Тестировать ПО на безопасность помогают программы, которые выявляют уязвимости в ПО:

* [NetSparker](https://www.netsparker.com/)
* [OWASP](https://owasp.org/)
* [Acunetix Vulnerability Scanner](https://www.acunetix.com/vulnerability-scanner/)

Нагрузочное тестирование является важнейшим элементом комплексного тестирования производительности, который направлен на анализ скорости реакции системы на внешний запрос. Такая проверка позволяет выяснять, отвечает ли испытуемое приложение либо устройство заявленным требованиям:

* [Apache JMeter](https://jmeter.apache.org/)
* [WebLOAD](https://www.radview.com/)
* [Micro-Focus Loadrunner](https://www.microfocus.com/ru-ru/products/loadrunner-professional/overview)

**2. КЛАССИФИКАЦИЯ ВИДОВ ТЕСТИРОВАНИЯ**

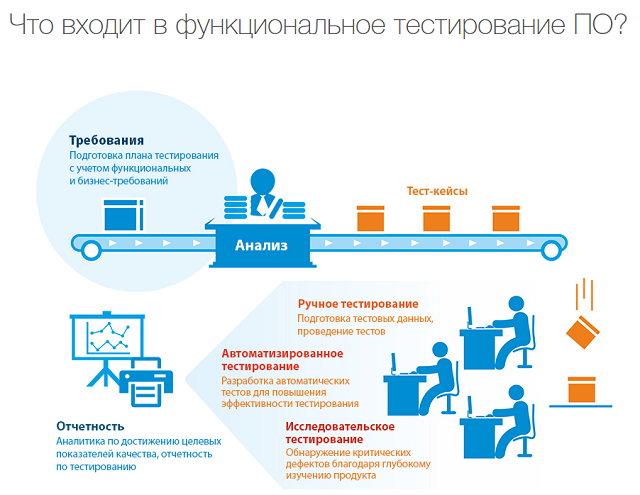
Каждый программный продукт должен выполнять одну или несколько ключевых задач. От приложения с гео-картами мы ожидаем точной ориентации в пространстве, от сайта интернет-магазина ― корректного поиска товаров по заданным параметрам и т. д. Но те же программные продукты мы можем протестировать и с точки зрения дизайна.

Таким образом, анализ ПО с позиции его ключевых или вспомогательных функций определяет тип тестирования:

**2.1. ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ**

* **Функциональное тестирование** (Functional testing) — Тестирование ПО, направленное на проверку того, что компонент либо система соответствует функциональным требованиям.
* **Функциональное требование (Functional requirement)** — Требование, определяющее функцию, которую компонент или система должны выполнять.
* **Функциональность (Functionality)** — Способность программного продукта обеспечивать функции, которые соответствуют установленным и предполагаемым потребностям, при использовании ПО в определенных условиях.

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



**2.2. НЕФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ**

**Нефункциональное** – проверка корректности работы нефункциональных требований. Оценивается, КАК программный продукт работает.

Эта проверка включает в себя следующие виды:

* **Тестирование графического интерфейса (GUI testing)** — Анализ соответствия графического пользовательского интерфейса программы спецификациям, макетам, прототипам, стандартам.
* **Тестирование удобства использования (Usability testing)** — Исследование, выполняемое с целью определения, удобна ли программа для ее предполагаемого применения и основанное на стандартах, лучших практиках и привлечении пользователей в качестве тестировщиков и суммировании и анализе полученных от них выводов.
* **Тестирование инсталляции (Installation testing)** — Тестирование, направленное на проверку процессов установки, удаления, восстановления, обновления, лицензирования.
* **Тестирование безопасности (Safety testing**) — Тестирование программного продукта с целью с целью определить его безопасность.
* ***Безопасность (Safety) —****Способность программного продукта при использовании оговоренным образом оставаться в рамках приемлемого риска причинения вреда здоровью, бизнесу, программам, собственности или окружающей среде.*
* **Тестирование защищенности (Security testing)**— Тестирование с целью оценить защищенность программного продукта.
* ***Защищенность (Security)****— Свойства программного продукта, отражающие его способность не допускать неавторизированный доступ, случайный или умышленный, к программам и данным.*
* **Тестирование доступности (Accessibility testing)** — Тестирование, направленное на определение степени легкости, с которой пользователи с ограниченными способностями могут использовать систему или ее компоненты.
* **Тестирование производительности (Performance testing)** — Процесс тестирования с целью определить производительность программного продукта.
* **Нагрузочное тестирование (Load testing)** — Вид тестирования производительности, проводимый с целью оценить поведение компонента или системы под увеличивающейся нагрузкой (число одновременно работающих пользователей и/или число транзакций) для определения максимально допустимого уровня нагрузки для исследуемого компонента или системы.
* **Стрессовое тестирование (Stress testing)** — Вид тестирования производительности, оценивающий систему или компонент на граничных значениях рабочих нагрузок или за их пределами, или же в состоянии ограниченных ресурсов, таких как память или доступ к серверу.
* **Тестирование интернационализации (Internationalization testing)** — Анализ приложения и его интерфейса на способность быть переведённым.
* **Тестирование локализации (Localization testing)** — Анализ приложения и его сопроводительной документации на корректность в конкретной переведённой среде.
* **Тестирование совместимости (Compatibility testing) -** нефункциональное тестирование для обеспечения удовлетворенности клиентов. Оно предназначено для определения того, может ли программное обеспечение или продукт работать в различных браузерах, базах данных, оборудовании, операционной системе, мобильных устройствах и сетях.
* **Тестирование кроссбраузерности (Cross browser testing)** — Вид тестирования совместимости, направленный на анализ работы приложения в разных браузерах либо разных версиях одного браузера.

**По степени автоматизации:**

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

* **Мануальное (manual)** – без использования дополнительных программных средств, т. е. «вручную».
* **Автоматизированное (avtomatization)** – с использованием программных средств (более детально в описании курса по автоматизации тестирования ПО).

Каждый из подходов имеет свои преимущества и недостатки. Ручное тестирование проще освоить, оно широко применяется на проектах всех типов, но мануальные проверки отличаются монотонностью. А вот написание тестов даёт больше возможностей для творческой реализации, но автоматизация требует базовых навыков программирования.

**По поведению системы:**

* **Позитивная проверка** – оценка ожидаемого поведения. Это тестирование проводится в первую очередь, ведь позволяет определить корректность работы программы.
* **Негативная** – определение устойчивости системы в нестандартной ситуации. Например, неожиданный сценарий взаимодействия пользователя с интерфейсом.

Эти типы тестирования нередко проводятся параллельно. Ведь работая над некоторой функциональностью, тестировщику проще оценить её поведение и в стандартных, и в нестандартных условиях.

**По доступу к коду:**

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»** – с доступом к коду.
* **Тестирование «черного ящика»** – без доступа к коду продукта.
* **Тестирование «серого ящика»** – на основе ограниченного знания внутренней структуры ПО. Часто говорят, что это смесь тестирования «белого ящика» и «чёрного 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.

Проверка программного продукта по каждому из сценариев требует достаточно глубоких знаний.

**По уровню детализации:**

* **Модульное / юнит-тестирование** – проверка корректной работы отдельных единиц ПО, модулей. Этот вид тестирования могут выполнять сами разработчики.
* **Интеграционное тестирование** – проверка взаимодействия между несколькими единицами ПО.
* **Системное** – проверка работы приложения целиком.
* **Приёмочное** – оценка соответствия заявленным требованиям к программному продукту.
* **Альфа-тестирование** – проверка программного продукта на поздней стадии разработки. Проводится разработчиками или тестировщиками.
* **Бета-тестирование** – оценка ПО перед выходом на рынок в фокус-группе или добровольцами. Отзывы собираются, анализируются и учитываются при внесении правок.

**По приоритету:**

* **Смоук тестирование (Smoke testing)** – проверка самой важной функциональности программного продукта.
* **Тестирование критического пути** **(Critical path testing)** – проверка функциональности, используемой типичными пользователями в повседневной деятельности.
* **Расширенное тестирование** **(Extended path testing)** – проверка всей заявленной функциональности.

**3. ПРИНЦИПЫ ТЕСТИРОВАНИЯ**

Разработка правильных и эффективных тестов – достаточно непростое занятие. Принципы тестирования, представленные ниже, были разработаны в последние 40 лет и являются общим руководством для тестирования в целом.

1. Тестирование может найти ошибки (Testing shows presence of defects). Тестирование может найти ошибки в ПО, но не доказать их отсутствие. Однако важно находить варианты тестов, которые будут выявлять как можно больше ошибок. Это позволит снизить вероятность появления ошибок в ПО. Ни в коем случае нельзя утверждать, что ПО не содержит ошибок даже если тестирование не выявило их.

2. Полное тестирование невозможно (Exhaustive testing is impossible). Нет возможности провести полное тестирование, включающее весь возможный ввод пользователя и состояния системы. Но необходимо правильно расставлять приоритеты и анализировать риски. Это может позволить более эффективно обеспечить качество ПО.

3. Раннее тестирование (Early testing). Тестирование необходимо начинать как можно раньше. На разных этапах жизненого цикла разработки ПО оно должно преследовать определенные цели.

4. Скопление дефектов (Early testing). Разные модули системы могут содержать разное количество дефектов, то есть плотность скопления дефектов в разных элементах программы может отличаться. Усилия по тестированию должны распределяться пропорционально фактической плотности дефектов. В основном, большую часть критических дефектов находят в ограниченном количестве модулей. Это проявление принципа Парето: 80% дефектов содержатся в 20% модулей.

5. Парадокс пестицида (Pesticide paradox). Прогоняя одни и те же тесты вновь и вновь, Вы столкнетесь с тем, что они находят все меньше новых ошибок. Поскольку ПО эволюционирует, многие из ранее найденных дефектов исправляют и старые тест-кейсы больше не срабатывают. Чтобы преодолеть этот парадокс, необходимо периодически вносить изменения в используемые наборы тестов, рецензировать и корректировать их с тем, чтобы они отвечали новому состоянию ПО и позволяли находить как можно большее количество дефектов.

6. Тестирование зависит от контекста (Testing is context dependent). Выбор методологии, техники и типа тестирования будет напрямую зависеть от природы самого ПО. Например, ПО для медицинских нужд требует гораздо более строгой и тщательной проверки, чем, например, сайт магазина. Из тех же соображений, сайт с большой посещаемостью должен пройти через серьезное тестирование производительности, чтобы показать возможность работы в условиях высокой нагрузки.

7. Заблуждение об отсутствии ошибок (Absence–of–errors fallacy). Тот факт, что тестирование не обнаружило дефектов, еще не значит, что ПО готово к публикации. Нахождение и исправление дефек- Проектирование и разработка тестов 29 тов будут не важны, если ПО окажется неудобным в использовании, и не будет удовлетворять ожиданиям и потребностям пользователя. 30 Структура документации тестирования

**4. ТЕХНИКИ ТЕСТ-ДИЗАЙНА**

#### **4.1. ТЕХНИКИ ТЕСТИРОВАНИЯ МЕТОДА ЧЁРНОГО ЯЩИКА:**

* **Эквивалентное разбитие (Equivalence partitioning)**

Идея тестирования по методу разбиения классов эквивалентности состоит в том, чтобы исключить набор входных данных, которые заставляют систему вести себя одинаково и давать одинаковый результат при тестировании программы.

Суть техники эквивалентного разбиения в том, чтобы:

- Разделить данные на группы (классы эквивалентности), которые, как предполагается, обрабатываются системой схожим образом (то есть ведут систему к одному состоянию);

- Из каждой группы (класса) выбрать одно значение и проверить его.

* **Анализ граничных значений (Boundary value analysis)**

Это техника проверки поведения продукта на крайних (граничных) значениях входных данных. Граничное тестирование также может включать тесты, проверяющие поведение системы на входных данных, выходящих за допустимый диапазон значений. При этом система должна определённым (заранее оговоренным) способом обрабатывать такие ситуации.

Граничные значения очень важны и их обязательно следует применять при написании тестов, т.к. именно в этом месте чаще всего и обнаруживаются ошибки.

На каждой границе диапазона следует проверить по три значения:

* граничное значение;
* значение перед границей;
* значение после границы.

Цель этой техники — найти ошибки, связанные с граничными значениями.

Алгоритм использования техники граничных значений:

* выделить классы эквивалентности;

(этот шаг является очень важным и от того, насколько правильным будет разбиение на классы эквивалентности, зависит эффективность тестов граничных значений);

* определить граничные значения этих классов;
* нужно понять, к какому классу будет относиться каждая граница;
* нужно провести тесты по проверке значения до границы, на границе и сразу после границы.
* **Анализ доменов (Domain analysis)**

Доменное тестирование — это техника тестирования метода чёрного ящика, направленная на уменьшение количества тестов путём одновременной проверки множества различных позитивных значений. Данная техника базируется на одновременном использовании техник эквивалентного разбиения и анализа граничных значений.

То есть суть доменного тестирования заключается в том, чтобы разделить набор условий тестирования на те значения, которые можно считать одинаковыми, и за счет этого протестировать эффективней.

* **Тестирование всех пар (All pairs testing, Pairwise)**

Попарное тестирование — это техника тест-дизайна, которая обеспечивает полное тестовое покрытие.

ISTQB определяет попарное тестирование как технику тест-дизайна при которой тест-кейсы создаются таким образом, чтобы выполнить все возможные отдельные комбинации каждой пары входных параметров.

Техника попарного тестирования очень помогает при разработке тестов для приложений, включающих множество параметров. Тесты разрабатываются таким образом, что для каждой пары входных параметров существуют все возможные комбинации этих параметров. Тестовые наборы (тест-сьюты, Test suite) охватывают все комбинации. Поэтому техника хоть и не обеспечивает исчерпывающее тестирование, но все же является эффективной для поиска ошибок.

* **Тестирование таблицы решений (Decision table testing)**

Таблица принятия решений — способ компактного представления модели со сложной логикой; инструмент для упорядочения сложных бизнес требований, которые должны быть реализованы в продукте. Это взаимосвязь между множеством условий и действий.

Эта техника основывается на принципе, что каждый тест-кейс должен проверять конкретный функциональный аспект приложения. Для достижения этой цели используется таблица принятия решений, которая позволяет разработчикам определить, какие варианты использования приложения следует проверять.

Таблица принятия решений представляет собой таблицу с двумя осями — вертикальной и горизонтальной. В вертикальной оси перечислены функциональные аспекты приложения, которые должны быть проверены, в то время как в горизонтальной оси перечислены различные варианты использования приложения. Каждый элемент таблицы содержит информацию о том, следует ли проверять соответствующий функциональный аспект для данного варианта использования приложения.

* **Тестирование диаграммы переходов состояний (State transition testing)**

Тестирование таблицы переходов – это метод, который используется там, где некоторый аспект системы может быть описан в так называемом «конечном автомате». Это просто означает, что система может находиться в (конечном) числе разных состояний, а переходы из одного состояния в другое определяются правилами «машины».

Схема состояний и переходов (от англ. State & Transition Diagram, S&T) — это схема переходов и состояния, специальная техника для перехода ТЗ из одного статуса в другой. С ее помощью пользователь в наглядной форме может просматривать переход продукта из одной стадии в другую.

Идеально подходит для длительных проектов, где техническое задание разбито на большие спринты, где требуется контроль и верификация любого действия.

Схематически подобную методику отображают в форме кругов и стрелочек, где:

Кружочки — это текущее состояние объекта;

Стрелочки — ситуация, событие или процесс, благодаря которым объект может двигаться из стадии А в стадию В. Это своего рода действие, которое может выполнятся как пользователем, так и системой. К примеру, процесс выполнения загрузки программы начался в 22:00 в автоматическом порядке.

Использование подобных схем позволяет наглядным образом визуально оценивать, какие переходы может выполнить ПО и что нужно протестировать в первую очередь. Стрелочки в данном случае — это уже сформированные тест-кейсы, которые и нужно проверять!

* **Тестирование сценариев использования (Use Case Testing)**

Тестирование вариантов использования — это метод идентификации тестовых сценариев, используемый для покрытия всей системы. Это помогает нам выявить пробелы в системе, из-за которых мы можем пропустить тестирование отдельных компонентов программного обеспечения.

Тестирование сценариев использования помогает анализировать систему на основе транзакций за транзакцией от начала до конца, что обеспечивает более широкий охват тестирования.

Это обычно используется для разработки тестов или систем на приемлемом уровне.

Типы вариантов использования

Сценарии использования обычно пишутся на описательном языке, описывающем функциональные требования. Но здесь есть две стороны медали: одна — функциональные требования с точки зрения пользователя, а другая — требования, которые должна выполнять система. Исходя из этого, существует два разных типа вариантов использования:

* Бизнес-вариант использования
* Системный вариант использования
* **Тестирование историй использования (User Story Testing)**

#### Что такое пользовательские истории (англ. user story)? Проще говоря, это описание требований к системе на языке пользователя (с его точки зрения). Вроде звучит понятно, но суть еще не ясна? Погрузимся поглубже.

#### Главная составляющая user story — это, собственно, сам юзер (user, пользователь). Здесь начинается самое интересное: нужно представить потребителя в виде персонажа, который будет взаимодействовать с нашим продуктом, прописать его потребности и боли, которые продукт закрывает.

пользовательская история — это описание функциональной возможности ПО простыми, общими словами, составленное с точки зрения конечного пользователя. Она пишется с целью разъяснить, как именно функциональная возможность принесет пользу клиенту.

Пользовательские истории — одна из базовых составляющих agile-программы. Они позволяют организовать повседневную работу в систему, ориентированную на пользователей, что способствует укреплению сотрудничества, поиску нестандартных идей и повышению качества продукта в целом

#### **4.2. ТЕХНИКИ ТЕСТИРОВАНИЯ МЕТОДА БЕЛОГО ЯЩИКА:**

* **Тестирование покрытия операторов (Statement testing and coverage)**

Целью этого метода тестирования является выполнение каждого оператора программы хотя бы один раз. Для фрагмента на рис. 2.1 можно выполнить каждый оператор, записав один единственный тест, который реализовал бы путь ace. Т.е., если бы на входе было: А=2, В=0, Х=3, каждый оператор выполнился бы один раз.

* **Тестирование покрытия условий и решений (Decision testing and coverage)**

Метод покрытия решений (покрытия переходов)

Согласно данному методу, каждое направление перехода должно быть реализовано по крайней мере один раз.

**Покрытие решений** обычно удовлетворяет критерию покрытия операторов. Поскольку каждый оператор лежит на некотором пути, исходящем либо из оператора перехода, либо из точки входа программы, при выполнении каждого направления перехода каждый оператор должен быть выполнен.

Для программы, приведенной на рисунке 2.1, покрытие решений может быть выполнено двумя тестами, покрывающими пути {ace, abd}, либо {aсd,abe}. Пути {aсd,abe}покроим, выбрав следующие исходные данные: {A=3, B=0, X=3} и {A=2, B=1, X=1}.

**Метод покрытия условий**

Лучшие результаты по сравнению с предыдущими методами может дать метод покрытия условий. В этом случае записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз.

В предыдущем примере имеем четыре условия: {A>1, B=0}, {A=2, X>1}. Для реализации метода требуется достаточное число тестов, такое, чтобы реализовать ситуации, где A>1, A£1, B=0 и B¹0 в точке а и A=2, A¹2, X>1 и X£1 в точке в. Тесты, удовлетворяющие критерию покрытия условий и соответствующие им пути:

A=2, B=0, X=4 ace

A=1, B=1, X=0 abd.

* **Тестирования покрытия условий и значений операторов (The value of statement and decision testing)**

Охват операторов — это метод проектирования теста белого ящика, который включает в себя выполнение всех исполняемых операторов в исходном коде как минимум один раз. Он используется для вычисления и измерения количества операторов в исходном коде, которые могут быть выполнены с учетом требований.

Охват операторов используется для выведения сценария на основе структуры тестируемого кода.

#### **4.3. ТЕХНИКИ, ОСНОВАННЫЕ НА ОПЫТЕ:**

* **Предугадывание ошибок (Error guessing)**

Предположение об ошибках – это способ предотвращения ошибок, дефектов и отказов, основанный на знаниях тестировщика, включающих:

— Историю работы приложения в прошлом.

— Наиболее вероятные типы дефектов, допускаемых при разработке.

— Типы дефектов, которые были обнаружены в схожих приложениях.

Структурированный подход к предположению об ошибках предполагает создание списка всех возможных ошибок, дефектов и отказов с последующей разработкой тестов, направленных на поиск дефектов из этого списка. Списки отказов и дефектов могут быть построены на основе опыта, исторических данных об отказах и ошибках, а также на общих знаниях о причинах отказа программ.

* **Исследовательское тестирование (Exploratory testing)**

**Простейшее определение исследовательского тестирования — это разработка и выполнения тестов в одно и то же время.**

Исследовательское тестирование объединяет этапы разработки и выполнения теста, обеспечивая полную свободу действий для тестировщика и позволяя ему постоянно оптимизировать свою работу.

Менее структурированный подход помогает тестировщикам определить, как приложение может отреагировать на реалистичные сценарии и тестовые примеры, и служит важным дополнением к сценарному тестированию.

Качество разведочного тестирования команды часто зависит от мастерства отдельных тестировщиков, поскольку проверки требуют творческого подхода и глубокого понимания программного обеспечения. Это процесс непрерывного открытия — процесс, в котором тестировщики используют дедуктивные рассуждения, чтобы направлять свою общую методику.

* **Свободное тестирование (Ad-hoc testing)**

Свободное/исследовательское тестирование (ad-hoc testing) - это вид тестирования, который выполняется без подготовки к тестированию продукта, без определения ожидаемых результатов, проектирования тестовых сценариев. Это неформальное, импровизационное тестирование. Оно не требует никакой документации, планирования, процессов, которых следует придерживаться при выполнении тестирования.

Такой способ тестирования в большинстве случаев дает большее количество заведенных отчетов об ошибке. Это обусловлено тем, что тестировщик на первых шагах приступает к тестированию основной функциональной части продукта и выполняет как позитивные, так и негативные варианты возможных сценариев.

Чаще всего такое тестирование выполняется, когда владелец продукта не обладает конкретными целями, проектной документацией и ранее поставленными задачами. При этом тестировщик полагается на свое общее представление о продукте, сравнение с похожими продуктами, собственный опыт. Однако при тестировании ad-hoc тестировщик должен иметь полные знания и осведомленность о тестируемой системе, особенно если проект очень сложный и большой. Поэтому нужно хорошее представление о целях проекта, его назначении, основных функциях и возможностях.

* **Тестирование по чек-листам (Checklist-based testing)**

Чек-листы тестировщика – это список задач, которые нужно выполнить в процессе тестирования. В чек-листе могут быть перечислены тест-кейсы, условия тестирования, требования к продукту и многое другое. Цель чек-листа – не пропустить ни одной важной детали в процессе тестирования.

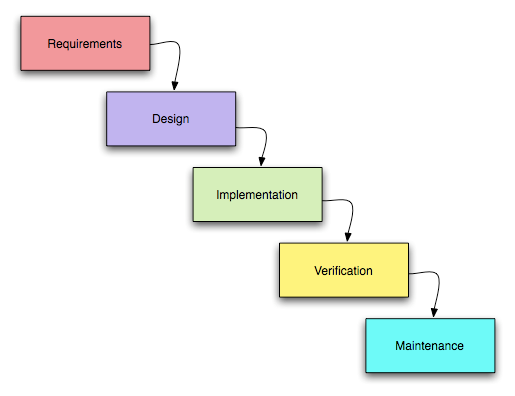
Использование чек-листов помогает тестировщикам упростить и структурировать свою работу, а также не забыть важные детали. Кроме того, чек-листы могут быть использованы в качестве инструмента для автоматизации тестирования.

Составление чек-листов – это процесс, который может значительно упростить и ускорить тестирование программного обеспечения.

**5. МОДЕЛИ ПРОЦЕССА РАЗРАБОТКИ ПО**

**5.1. WATERFALL**

**Каскадная модель** (англ. waterfall model, иногда переводят, как модель «Водопад») — модель процесса разработки программного обеспечения, в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки.

[](http://qaevolution.ru/wp-content/uploads/2016/01/2.png)Следуя каскадной модели, разработчик переходит от одной стадии к другой **строго последовательно**. Сначала полностью завершается этап «определение требований», в результате чего получается список требований к ПО. После того как требования полностью определены, происходит переход к проектированию, в ходе которого создаются документы, подробно описывающие для программистов способ и план реализации указанных требований. После того как проектирование полностью выполнено, программистами выполняется реализация полученного проекта. На следующей стадии процесса происходит интеграция отдельных компонентов, разрабатываемых различными командами программистов. После того как реализация и интеграция завершены, производится тестирование и отладка продукта; на этой стадии устраняются все недочёты, появившиеся на предыдущих стадиях разработки. После этого программный продукт внедряется и обеспечивается его поддержка — внесение новой функциональности и устранение ошибок.

**5.2. AGILE**

В современной разработке наибольшей популярностью пользуется целое семейство методологий — **Agile.**

Agile, или Agile software development, — это гибкий подход [к управлению проектами](https://practicum.yandex.ru/blog/chto-takoe-upravlenie-proektami/) по разработке программного обеспечения (ПО), который часто применяют в небольших командах.

Как правило, для гибкого подхода Agile характерна работа короткими итерациями по две-три недели. Внутри каждой итерации собрана серия задач: анализ, проектирование, непосредственно работа и тестирование. После каждой итерации команда анализирует результаты и меняет приоритеты для следующего цикла.

Подход Agile возник после того, как в сфере IT устали от излишней бюрократии и строгости. Разработчики поняли, что создавать инновационные продукты по старым строгим методологиям просто нельзя, поэтому в 2001 году в американском штате Юта 17 разработчиков со всего света собрались и подписали манифест о новых передовых принципах разработки, которые и легли в основу Agile.



**Принципы Agile**

1. Удовлетворение клиентов — приоритетная задача при разработке продукта. Клиенты должны своевременно и в полном объёме получать качественное программное обеспечение и его обновления.
2. Изменения в процессе разработки приветствуются. Гибкие процессы позволяют наделить продукт конкурентными преимуществами для клиентов.
3. Рабочее ПО нужно доставлять клиенту часто, в рамках 2–16 недель.
4. Руководители и разработчики должны трудиться вместе на протяжении всего рабочего процесса.
5. В основе проекта — мотивированные люди. Обеспечьте им необходимые условия работы, поддержку и доверие.
6. Лучший способ передачи информации в команде — личная беседа.
7. Основное мерило прогресса — работающее ПО. А не часы, трудозатраты и другие критерии.
8. Гибкие процессы — основа устойчивого развития. Они позволяют поддерживать нужный рабочий темп как на спринтерской, так и на марафонской дистанции.
9. Важно уделять внимание техническому совершенству и качественному дизайну продукта.
10. Важно сокращать до минимума лишнюю работу и не переусложнять проект и рабочие процессы.
11. Самые лучшие продукты рождаются у самоорганизующихся команд. Нет микроменеджменту, да — свободе управления.
12. Команда должна регулярно оценивать работу и корректировать своё поведение.

В семейство Agile входит несколько разных гибких [методологий управления проектами](https://practicum.yandex.ru/blog/metodologii-upravleniya-proektami/), которые ещё называют методами управления. В России наибольшей популярностью пользуются две — Scrum и Kanban.

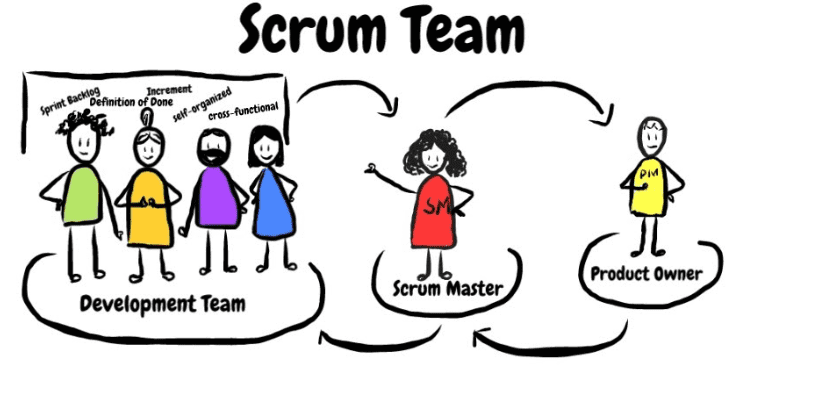
# **5.2.1. Scrum**

**Скрам (Scrum**) — очень модное на сегодняшний день слово и одна из самых популярных Agile техник. Огромное количество проектов пытается использовать этот фреймворк, но, к сожалению, не у всех есть понимание того, как он должен работать.

Скрам (Scrum) — итеративно-инкрементальный, гибкий (Agile) фреймворк (каркас разработки), позволяющий в процессе постоянной коммуникации за короткое время, называемое спринтом, доставлять инкремент продукта.

*Почему Scrum — это не методология?*

В скраме нет чётко описанных процессов, этапов, методов разработки и тестирования. То есть нет чёткого описания того, как всё должно работать.

В свою очередь, скрам описывает лишь роли и некоторые правила игры, которым нужно следовать. При этом остаётся простор для фантазии и экспериментов, чтобы сделать процесс лучше.

Команда в скраме (Scrum Team) обычно не более 10 человек и состоит из 3 основных ролей:

* Product Owner
* Scrum Master
* Development team

Скрам очень гибкий инструмент, который нацелен на постоянное улучшение качества процесса и продукта.

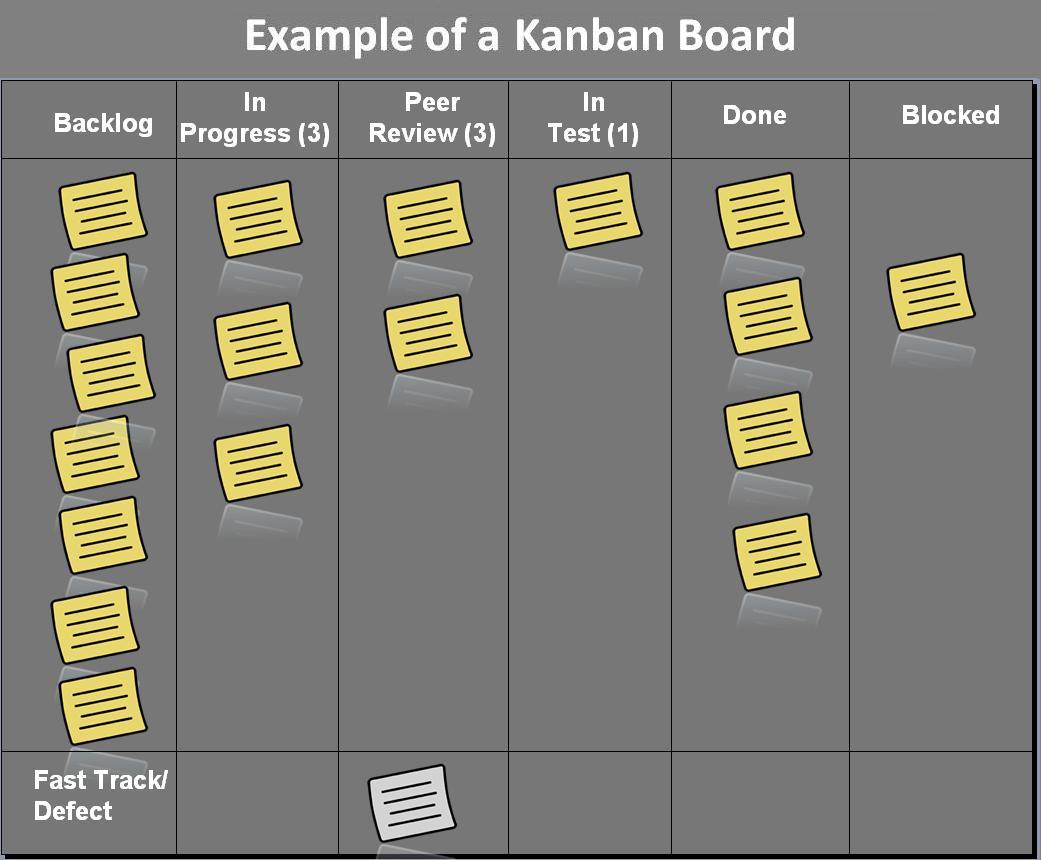
Этот фреймворк предоставляет каркас, который можно дополнять и улучшать никак себя не ограничивая. Но отказ от какой-то роли или процесса автоматически лишает Вас статуса Scrum.

**5.2.2. KANBAN**

**Kanban** — одна из популярных гибких (Agile) техник, которая отлично подходит для визуализации и оптимизации потока задач. Но, к сожалению, у начинающих тестировщиков этот метод вызывает огромное количество проблем и вопросов. В этой статье мы немного погрузимся в Kanban и разберём то, как он работает.

**Отличительные черты:**

1. Прозрачность процесса путём визуализации разработки с помощью Kanban доски.
2. Ограничение количества задач, находящихся на одной стадии (Work In Progress, WIP), например максимум 5 задач может находиться одновременно на этапе разработки

Ниже можно увидеть пример **Kanban**доски с ограничениями.

На стадиях**In Progress, Peer Review, In Test** одновременно может находиться не более того числа, что находится в скобках.

*Для чего это нужно?*

Для того, чтобы команда не делала параллельно несколько задач, а старалась как можно быстрее довести одну задачу из In Progress в Done. Тем самым уменьшив среднее время доставки одной задачи в Done.

Kanban лучше всего подходит для проектов, где необходим потоковый процесс и непрерывный деплоймент на продакшн окружение. Например, саппорт или любой проект, где каждое изменение после тестирования сразу попадает в продакшн.

Kanban — одна из самых неформальных Agile техник, т.к., по сути, есть только 2 правила — визуализация и ограничение WIP.

Цель Kanban — сделать так, чтобы в среднем одна задача выполнялась как можно быстрее.

Итеративная методология с доской, но без ограничения WIP — это не Kanban.

Kanban отлично подходит для проектов, где должна быть реализована непрерывная поставка изменений на продакшн.

# **6. АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ**

**6.1. АЛГОРИТМ ДЕЙСТВИЙ**

Автоматизация тестирования – это процесс использования программных средств для выполнения тестовых сценариев. Этот процесс это [один из видов тестирования](https://qaevolution.ru/testirovanie-po/vidy-testirovaniya-po/) позволяет улучшить качество и скорость тестирования, а также сократить затраты на тестирование.

Автоматизация тестирования позволяет улучшить качество и скорость тестирования, а также сократить затраты на тестирование. Когда тестирование выполняется вручную, это может быть очень трудоемким процессом. Например, если тестирование включает множество шагов, которые необходимо повторить несколько раз, это может занять много времени и сил. Кроме того, при ручном тестировании возникает риск ошибок, так как человек может пропустить какую-то деталь или не заметить ошибку. Автоматизация позволяет сократить время тестирования, увеличить точность тестирования и уменьшить риск ошибок.

Следующие типы тестовых сценариев полезно автоматизировать:

* Тесты, которые необходимо повторять несколько раз
* Тесты, которые сложны для выполнения вручную
* Тесты, которые занимают много времени
* Тесты, которые требуют большого объема входных данных
* Тесты, которые требуют быстрого выявления ошибок

## Автоматизация тестирования алгоритм действий

Алгоритм действий при автоматизации тестирования может быть следующим:

1. Определение целей: определение задач, которые должны быть автоматизированы, и оценка возможных выгод от автоматизации.
2. Выбор инструментария: выбор инструментов, которые будут использоваться для автоматизации тестирования. Он зависит от особенностей тестируемого приложения и технических возможностей команды тестировщиков.
3. Создание тестовых сценариев: создание тестовых сценариев для автоматизации, учитывая особенности приложения и требования к тестированию.
4. Разработка тестовых скриптов: создание скриптов для автоматизации выполнения тестовых сценариев. В этом процессе тестировщик создает скрипт на выбранном инструменте, который будет эмулировать действия пользователя в приложении.
5. Настройка окружения: настройка окружения тестирования, включая установку необходимых программных компонентов, баз данных, настройку серверов и т.д.
6. Запуск тестовых скриптов: запуск тестовых скриптов в автоматическом режиме и сбор результатов.
7. Анализ результатов: анализ результатов тестирования и сравнение их с ожидаемыми результатами. В случае обнаружения ошибок или несоответствий, тестировщик должен создать отчет о баге и отправить его разработчикам для исправления.
8. Поддержка и обновление: поддержка и обновление автоматизированных тестов при изменении приложения или его функциональности.

## Рекомендации для эффективной автоматизации

1. Определите, какие тест-кейсы будут автоматизированы. Не все тест-кейсы должны быть автоматизированы, поэтому необходимо выбрать те тесты, которые требуют больше времени и усилий при ручном тестировании, или тесты, которые необходимо повторять многократно.
2. Выберите подходящий инструмент для автоматизации. Существует множество инструментов для этого процесса, каждый из которых имеет свои преимущества и недостатки. Необходимо выбрать инструмент, который лучше всего соответствует требованиям проекта.
3. Создайте хорошо организованную структуру тестов. Эффективная структура тестов поможет избежать дублирования тестов и сократить время на разработку тест-кейсов. Тесты должны быть легко понятны и доступны для всех членов команды.
4. Используйте модульное тестирование. Модульное тестирование позволяет тестировать каждый модуль отдельно, что упрощает процесс отладки и позволяет избежать проблем, связанных с зависимостью модулей друг от друга.
5. Напишите надежный код тестов. Код тестов должен быть написан таким образом, чтобы он был легко понятен и изменяем. Код тестов должен быть также хорошо организован и поддерживаем.

## Особенности автоматизированного тестирования

Автоматизированное тестирование имеет свои особенности, которые следует учитывать при использовании этой методики. Вот некоторые из них:

1. Необходимость постоянного обновления тестовых скриптов: любое изменение в приложении может привести к тому, что ранее написанные тесты станут нерелевантными. Поэтому тестовые скрипты должны быть постоянно обновляемыми, чтобы отражать текущее состояние приложения.
2. Необходимость высокой степени автоматизации: для того чтобы автоматизация была эффективной, необходимо автоматизировать как можно большее количество тестов. Это позволяет существенно сократить время на тестирование и увеличить покрытие тестами.
3. Необходимость подготовки тестовых данных: для успешного автоматизированного тестирования необходимо создать тестовые данные, которые будут использоваться в тестах. Это может быть трудоемкой задачей, особенно если приложение имеет сложную структуру данных.
4. Необходимость правильного выбора инструментов: выбор инструментов для автоматизации тестирования является очень важным. Необходимо выбирать инструменты, которые подходят для конкретного проекта и которые имеют необходимую функциональность.
5. Необходимость обучения: автоматизация тестирования требует знаний и опыта в области программирования и тестирования. Поэтому необходимо обучить тестировщиков, которые будут работать с автоматизированными тестами.

## 6.2. ИНСТРУМЕНТЫ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ

Существует множество инструментов для автоматизации тестирования, открытых и закрытых, бесплатных и платных. Вот некоторые из них:

1. Selenium — наиболее популярный инструмент для автоматизации веб-приложений. Selenium можно использовать для написания тестов на многих языках программирования, включая Java, Python, C# и Ruby.
2. Appium — инструмент для автоматизации тестирования мобильных приложений на платформах Android и iOS. Appium использует WebDriver протокол для автоматизации действий на устройствах.
3. JMeter — инструмент для тестирования производительности и нагрузочного тестирования веб-приложений. JMeter может использоваться для тестирования различных протоколов, включая HTTP, HTTPS, FTP, SMTP и JDBC.
4. TestComplete — инструмент для автоматизации тестирования десктопных, мобильных и веб-приложений. TestComplete предоставляет множество готовых модулей для автоматизации тестирования, а также позволяет написать свои собственные скрипты на языках JavaScript, Python и VBScript.
5. Robot Framework — фреймворк для автоматизации тестирования, позволяющий писать тесты на многих языках программирования, включая Python, Java, .NET и Perl. Robot Framework поддерживает различные протоколы, такие как HTTP, FTP, SSH, JDBC и другие.
6. Cucumber — инструмент для автоматизированного тестирования приложений на основе поведения. Cucumber позволяет писать тесты на естественном языке, который понятен как разработчикам, так и тестировщикам.
7. Postman — инструмент для тестирования API и создания запросов. Postman позволяет тестировать API без написания кода, а также автоматизировать тесты с помощью JavaScript.
8. SoapUI — инструмент для автоматизации тестирования веб-сервисов на основе протокола SOAP. SoapUI поддерживает различные протоколы, такие как HTTP, JMS, AMF и JDBC.
9. LoadRunner — инструмент для тестирования производительности и нагрузочного тестирования веб-приложений. LoadRunner поддерживает множество протоколов, включая HTTP, FTP, SMTP и LDAP.
10. Katalon Studio — инструмент для автоматизации тестирования веб-приложений, мобильных приложений и API. Katalon Studio предоставляет графический интерфейс

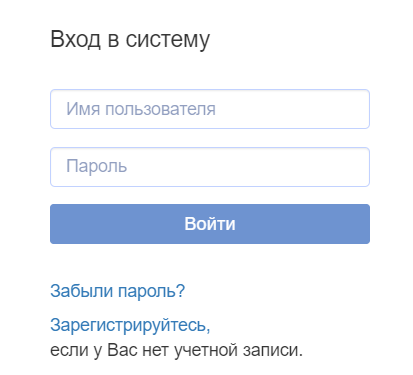
**ТЕХНИЧЕСКАЯ ДОКУМЕНТАЦИЯ**

МОДУЛЬ «ЛИЧНЫЙ КАБИНЕТ ПОЛЬЗОВАТЕЛЯ»

(ВХОД В СИСТЕМУ ЗАРЕГИСТРИРОВАННОГО ПОЛЬЗОВАТЕЛЯ,

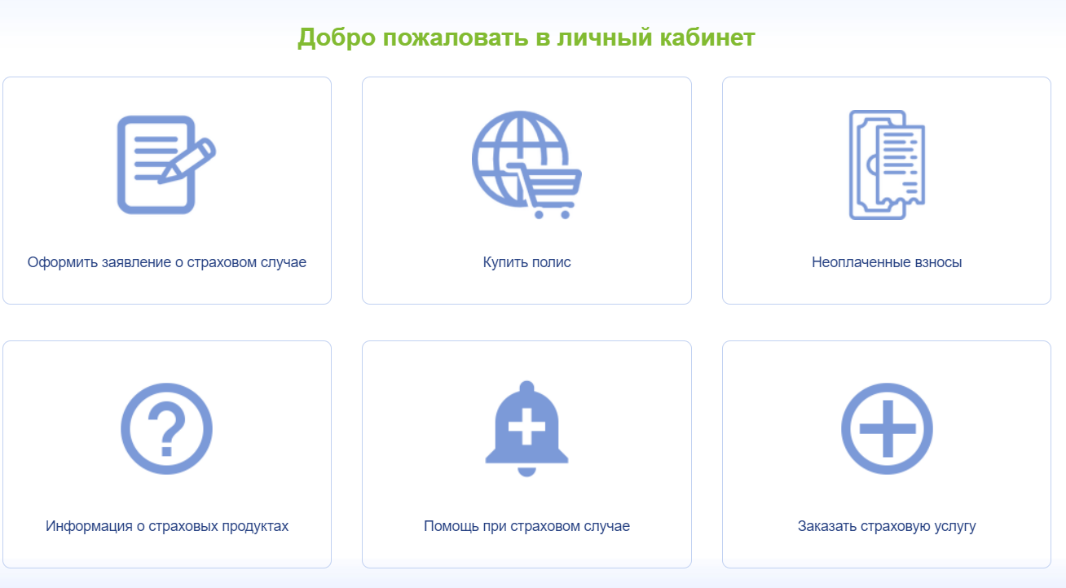
РЕГИСТРАЦИЯ НОВОГО ПОЛЬЗОВАТЕЛЯ)

При клике на значок «Личный кабинет пользователя» отображается форма для входа в систему **зарегистрированного пользователя:**

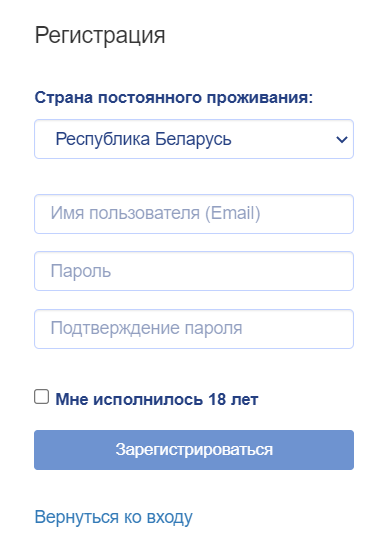
**В поле «Имя пользователя (Email)»** допускается ввод электронной почты в формате [your@email.com](mailto:your@email.com) (в первой части допускается ввод спецсимволов и цифр, т.к. электронный адрес может их содержать). Поля содержат плейсхолдеры «имя пользователя (email)», «пароль».

При вводе невалидной почты при клике на «Зарегистрироваться» появляется тултип с ошибкой красного цвета «Ошибка при проверке данных пользователя. Неверное имя пользователя или пароль». Максимально допустимая длина поля 30 символов.

**В поле «Пароль»** допускается ввод цифр, букв (строчные и заглавные), спецсимволов. Минимальное значение поля - 4 символа, максимальное - 20 символов.

После клика на кнопку «Войти» происходит переадресация в личный кабинет.

**Если пользователь не зарегистрирован**, необходимо кликнуть на надпись «Зарегистрируйтесь». Далее для успешной регистрации необходимо выбрать из выпадающего списка страну постоянного проживания, заполнить обязательные поля (Имя пользователя (Email) и пароль).



**В поле «Имя пользователя (Email)»** допускается ввод электронной почты в формате [your@email.com](mailto:your@email.com) (в первой части допускается ввод спецсимволов и цифр, т.к. электронный адрес может их содержать). При вводе невалидной почты при клике на «Зарегистрироваться» появляется текст с ошибкой красного цвета. Максимально допустимая длина поля 30 символов.

**В поле «Пароль» и «Подтверждение пароля»** допускается ввод цифр, букв (строчные и заглавные), спецсимволов. Минимальное значение поля - 4 символа, максимальное - 20 символов.

Для успешной регистрации кликнуть на чек-бокс «**Мне исполнилось 18 лет».**

При клике «Зарегистрироваться» происходит регистрация нового пользователя.

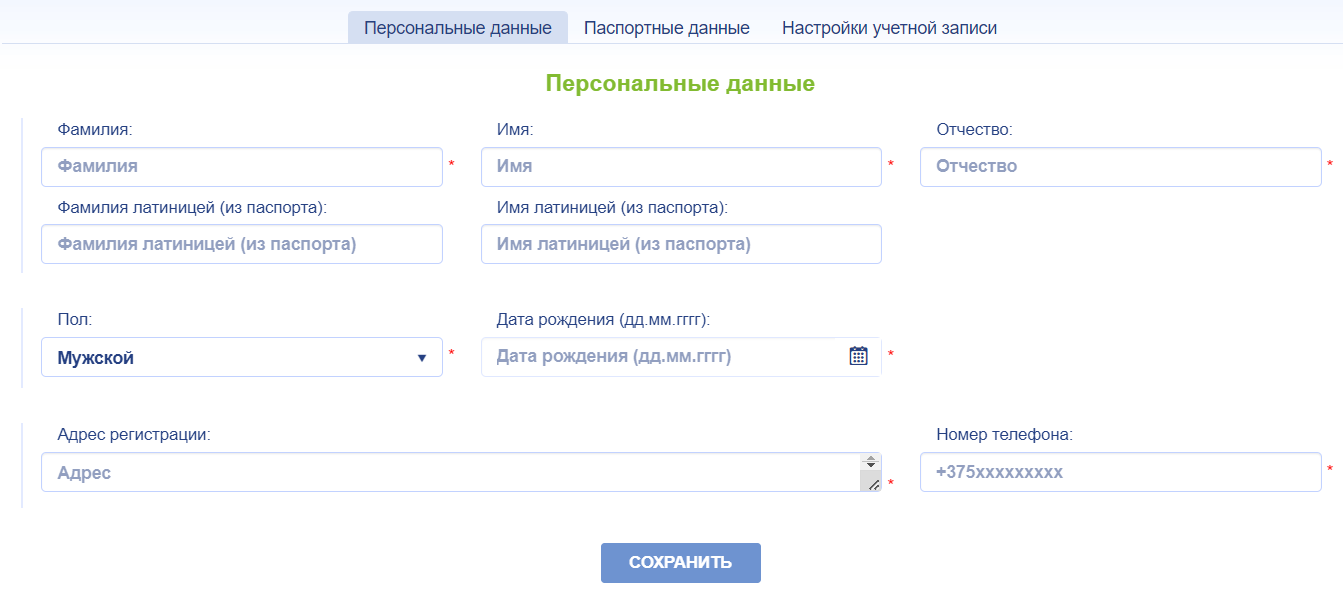
Если пользователь оставит чек-бокс неактивным, выше появится сообщение: «Лица моложе 18 лет не могут выступать страхователями по договору страхования».

После регистрации происходит переадресация в личный кабинет, но для его активации необходимо ввести Персональные данные пользователя. Для этого вверху справа, возле электронного адреса пользователя, нажать на .

Появится новая форма для заполнения с тремя вкладками «Персональные данные пользователя», «Паспортные данные», «Настройка учетной записи». Наименование вкладок отображается – шрифт 14рх, цвет #254082.

Если вкладка активна, она подсвечивается синим цветом. Необходимо заполнить обязательные поля, помеченные звездочкой: фамилия, имя, отчество, пол, дата рождения, адрес регистрации, номер телефона.

Наименование самой вкладки отображается – шрифт 18,6667рх, цвет #82ВВ31.

**В поля «Фамилия», «Имя», «Отчество»** можно ввести только текстовые значения. Допускается кириллица. Максимальная длина полей 30 символов. Данные поля содержат плейсхолдер.

**В поля «Фамилия латиницей (из паспорта)» и «Имя латиницей (из паспорта)»** можно ввести только текстовые значения. Допускается латиница. Данные поля содержат плейсхолдер.

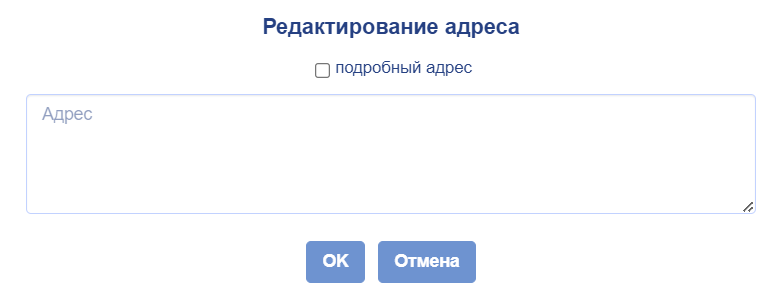
**В поле «Пол»** необходимо выбрать из выпадающего списка пол пользователя.

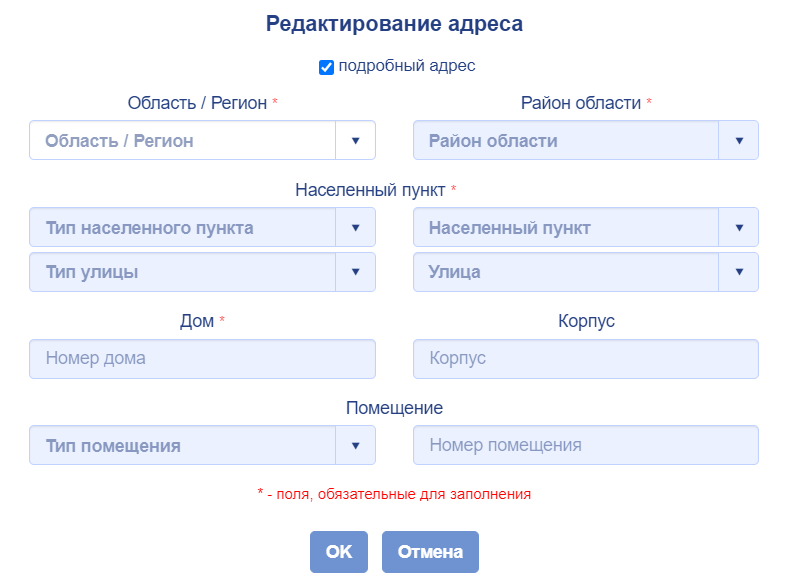
**При клике на поле «Дата рождения»** открывается календарь для выбора даты рождения. Выбранная дата отражается в формате дд.мм.гггг. Есть возможность ввести дату вручную в формате дд.мм.гггг. Для возможности регистрации на портале пользователю должно быть не менее 18 лет. Даты рождения, то которых на сегодня прошло менее 18 лет, выбрать нельзя.

**В поле «Номер телефона»**, в форме для заполнения «Персональные данные пользователя», необходимо ввести + в самом начале и 12 цифр. Ввод других символов недопустим – система их не принимает. Ввод цифр длиной больше 12 символов также не допустим. Если пользователь пытается отправить форму с введенным телефоном неверного формата (нет плюса, менее 12 цифр), то система выдает тултип с ошибкой: Во вкладке "Персональные данные" поле "Телефон" обязательно должно содержать цифры, также в начале используется один символ "+". Другие символы не допускаются»

Если пользователь пытается отправить форму с не введенным номером телефона, система выдает тултип с ошибкой: «Во вкладке "Персональные данные" не заполнено поле "Телефон"»

**При клике на поле «Адрес регистрации»** появляется форма «Редактирование адреса» с отражением чек-бокса «Подробный адрес» и формы заполнения.

**При неактивном чек-боксе «Подробный адрес»** необходимо вручную ввести адрес регистрации в поле заполнения. Допускается ввод букв, цифр, спецсимволов.

**При активном чек-боксе «Подробный адрес»** необходимо заполнить обязательные поля в форму заполнения: область/регион, район области, населенный пункт (тип местности, улица), дом.

**В поле «Область/Регион»** из выпадающего списка выбрать область/регион.

**В поле «Район области»** из выпадающего списка выбрать район.

**В полях «Населенный пункт»** выбрать из выпадающего списка: тип населенного пункта, наименование населенного пункта, тип улицы. В поле названия улицы ввести наименование улицы. При введении первой буквы появляется выпадающий список улиц населенного пункта.

**В поле «Дом»** необходимо ввести номер дома. Допускаются буквы, цифры и спецсимволы.

**В поле «Корпус»** необходимо ввести номер дома. Допускаются буквы, цифры и спецсимволы.

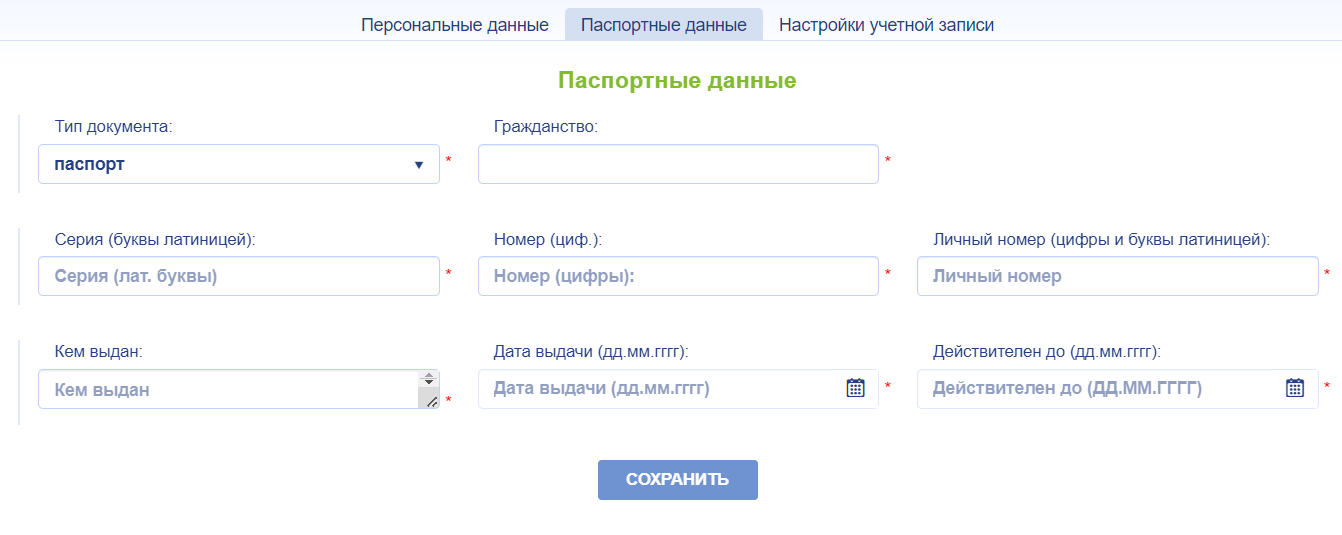
**В поле «Помещение»** необходимо выбрать из выпадающего списка тип помещения.

**В поле «Номер помещения»** необходимо ввести номер помещения. Допускаются буквы, цифры и спецсимволы.

При не заполнении обязательных полей, ниже появляется мигающее, красного цвета, сообщение: «Невозможно сохранить адрес, т.к. не заполнены все обязательные поля».

При клике на кнопку «ОК» происходит редактирование адреса и перенаправление на форму «Паспортные данные». Вкладка «Паспортные данные» подсвечивается синим.

Если в форме «Персональные данные» не заполнено какое-либо поле, система выдает ошибку: Во вкладке "Персональные данные" не заполнено поле "Фамилия"

ВКЛАДКА ПАСПОРТНЫЕ ДАННЫЕ

**В поле «Тип документа»** выбираем из выпадающего списка тип документа.

**При клике на поле «Гражданство»** появляется выпадающий список стран с возможность скролла и выбора страны, также страну можно ввести вручную. При вводе первой буквы страны, появляется выпадающий список стран на эту букву.

**В поле «Серия (буквы латиницей)»** необходимо ввести серию документа. Допускаются только латинские буквы (строчные и заглавные). Поле содержит плейсхолдер. При вводе не валидных значений система выдает ошибку: Во вкладке "Паспортные данные" поле "Серия" заполнено недопустимыми символами (необходимо использовать символы латинского алфавита).

**В поле «Номер (циф.)»** необходимо ввести номер документа. Максимальная длина - 7 символов. Допускаются только цифры. Ввод иных символов невозможен. Поле содержит плейсхолдер.

**В поле «Личный номер»** необходимо ввести личный номер документа. Максимальная длина - 14 символов. Допускаются только буквы латинского алфавита (строчные и заглавные) и цифры. Поле содержит плейсхолдер. При вводе не валидных значений система выдает ошибку: Во вкладке "Паспортные данные" поле "Личный номер" заполнено недопустимыми символами (необходимо использовать символы латинского алфавита).

**В поле «Кем выдан»** необходимо ввести данные кем выдан документ. Максимальная длина - 50 символов. Допускаются буквы (кириллица), цифры, спецсимволы, пробел. Поле содержит плейсхолдер.

**При клике на поле «Дата выдачи (дд.мм.гггг)»** открывается календарь для выбора даты. Выбранная дата отражается в формате дд.мм.гггг. Есть возможность ввести дату вручную в формате дд.мм.гггг. Допускается ввод цифр, спецсимволов.

**При клике на поле «Действителен до (дд.мм.гггг)»** открывается календарь для выбора даты. Выбранная дата отражается в формате дд.мм.гггг. Есть возможность ввести дату вручную в формате дд.мм.гггг. Допускается ввод цифр, спецсимволов.

При клике на кнопку «Сохранить» появляется тултип «**Соглашение об использовании персональных данных клиентов Белгосстраха»** с чек-боксом **«Я ознакомился с соглашением и принимаю условия».** При активном чек-боксе, кнопка «ОК» становится кликабельна. После клика на кнопку «ОК» система выдает сообщение: Данные успешно сохранены!.

**СПИСОК ЛИТЕРАТУРЫ**

1. Автоматизированное тестирование [Электронный ресурс]/ GitHub. URL: https://gist.github.com/codedokode/a455bde7d0748c0a351a

2. Виды Тестирования [Электронный ресурс]/ Про Тестинг - Тестирование Программного Обеспечения. URL: <http://www.protesting.ru/testing/types/sanity.html>

3. Канер С., Фолк Дж., Енг Кек Нгуен. Тестирование программного обеспечения: Фундаментальные концепции менеджмента бизнес-приложений

4. Котляров В. П., Коликова Т. В. Основы тестирования программного обеспечения

5. Гленфорд Майерс, Том Баджетт, Кори Сандлер “Искусство тестирования программ”

6. Д. Уиттакер, Д. Арбон, Д. Каролло «Как тестируют в Google»

7. Основные положения тестирования [Электронный ресурс]/ Интересные публикации / Хабрахабр. URL: <https://habrahabr.ru/post/110307/>

8. Рекс Блек. Ключевые процессы тестирования - М.: Издательство Лори, 2014. - 544 с.

9. Савин Роман. Тестирование DOT COM. Издательство Дело, 2007.

10. С. Куликов «Тестирование программного обеспечения. Базовый курс»

11. Тамре Л. Введение в тестирование программного обеспечения