Министерство образования и науки Российской федерации

Федеральное государственное автономной образовательное учреждение высшего образования

**«Уральский федеральный университет**

**имени первого Президента России Б.Н. Ельцина»**

**Институт новых материалов и технологий**

**Инженерная школа новой индустрии**

Дружинин Михаил Вадимович

ПРОЕКТИРОВАНИЕ СИСТЕМЫ **ВЕРИФИКАЦИИ** ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ **С ПРИМЕНЕНИЕМ СИСТЕМНОЙ ИНЖЕНЕРИИ**

27.04.03 «Системный анализ и управление»

**Диссертация на соискание академической**

**степени магистра**

Научный руководитель, к.т.н., доцент Мизгулин В. В.

Нормоконтролер Кравченко Н. С.

Студент гр. НМТМ-262701 Дружинин М. В.

Екатеринбург

2018

задание

# РЕФЕРАТ

Выпускная квалификационная работа 86 с., 39 рис., 14 табл., 8 источников.

ПРОЕКТИРОВАНИЕ СИСТЕМЫ ВЕРИФИКАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ПРИМЕНЕНИМ СИСТЕМНОЙ ИНЖЕНЕРИИ

Объектом проектирования является система верификации программного обеспечения.

Цель работы - спроектировать комплекс мероприятий для контроля качества в рамках процесса разработки ПО с помощью системного подхода.

Задачи работы

1. Рассмотреть основные понятия, касающиеся проблемы исследования: качество в процессе разработки ПО, контроль качества, обеспечение качества.
2. Изучить процесс разработки ПО и выделить основные аспекты, влияющие на качество.
3. Выявить инструмент для осуществления тестирования ПО.
4. Разработать практические рекомендации по усовершенствованию процесса разработки ПО.

В ходе работы были применены методики системного анализа, реализованы компетенции по проектированию процесса тестирования. Применены ментальные карты, подходы заполнению и ведению-тест плана, организация работы с автоматизацией тестирования, создание тестовой отчетности и ее доставка до стейкхолдеров.

Выполнено системно-инженерное описание проекта. Спроектирована система верификации программного обеспечения. Создана техническая документация, достаточная для реализации инструмента и апробация системы в процессе разработки программного продукта «ZOZO WFM»

Содержание

[РЕФЕРАТ 3](#_Toc516880258)

[ВВЕДЕНИЕ 6](#_Toc516880259)

[ГЛАВА 1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ КОНТРОЛЯ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 10](#_Toc516880260)

[1.1 Описание жизненного цикла разработки программного обеспечения 10](#_Toc516880261)

[1.2 Основные характеристика качества 12](#_Toc516880262)

[1.3 Основные характеристики контроля качества 14](#_Toc516880263)

[1.4 Обеспечение качества 15](#_Toc516880264)

[1.5 Обзор существующих подходов к контролю качества 15](#_Toc516880265)

[2.1 Характеристика деятельности системы «ZOZO WFM» 18](#_Toc516880266)

[2.2 Использование методологии «черный ящик» 19](#_Toc516880267)

[2.3 Определение операционного контекста проектируемой системы 20](#_Toc516880268)

[2.3.1 Функция «Обеспечить работу с клиентом» 21](#_Toc516880269)

[2.3.2 Функция «Обеспечить проработку требований» 21](#_Toc516880270)

[2.3.3 Функция «Обеспечить инфраструктуру» 22](#_Toc516880271)

[2.3.4 Функция «Обеспечить написание кода» 22](#_Toc516880272)

[2.3.5 Функция «Обеспечить приемку результата» 22](#_Toc516880273)

[2.3.6 Функция «Обеспечить контроль качества» 23](#_Toc516880274)

[2.4 Определение стейкхолдеров системы «ZOZO WFM» 23](#_Toc516880275)

[2.4.1 Заказчик системы «ZOZO WFM» 25](#_Toc516880276)

[2.4.2 Инвестор проекта «ZOZO WFM» 25](#_Toc516880277)

[2.4.3 Пользователь системы «ZOZO WFM» 26](#_Toc516880278)

[2.4.4 Аналитик 26](#_Toc516880279)

[2.4.5 Инженер администрирования 26](#_Toc516880280)

[2.4.6 Разработчик программного обеспечения 27](#_Toc516880281)

[2.5 Моделе-ориентированные бизнес-анализ 27](#_Toc516880282)

[2.5.1 Функциональный бизнес-контекст 27](#_Toc516880283)

[2.5.2 Структурная модель 30](#_Toc516880284)

[2.5.3 Модель стейкхолдеров 32](#_Toc516880285)

[2.5.4 Определение целей стейкхолдеров 32](#_Toc516880286)

[2.6 Моделе-ориентированная работа со стейкхолдерами 33](#_Toc516880287)

[2.6.1 Функциональная модель системы контроля качества в бизнес-контексте 33](#_Toc516880288)

[2.6.2 Определение требований стейкхолдеров 35](#_Toc516880289)

[2.6.3 Сопоставление требований стейкхолдеров 37](#_Toc516880290)

[2.7 Выбор инструмента тестирования 39](#_Toc516880291)

[2.7.1 Детализация показателей 39](#_Toc516880292)

[2.7.2 Описание и оценки альтернатив 42](#_Toc516880293)

[2.7.3 Анализ эффективности, рисов, перспектив развития 48](#_Toc516880294)

[2.7.4 Рекомендации по выбору инструмента 51](#_Toc516880295)

[ГЛАВА 3 ПРОЕКТИРОВАНИЕ СИСТЕМЫ ВЕРИФИКАЦИИ 52](#_Toc516880296)

[3.1 Разработка подхода к описанию тест-плана 52](#_Toc516880297)

[3.2 Формула заполнения тест-плана 55](#_Toc516880298)

[3.2.1 Требования к заполнению поле «Действие» 55](#_Toc516880299)

[3.2.2 Дополнения к заполнению поля «Действие» 56](#_Toc516880300)

[3.3 Использование ментальных карт 56](#_Toc516880301)

[3.4 Практическое применение инструмента «Selenium» 60](#_Toc516880302)

[3.4.1 Инструкция для работы с «Selenium» 60](#_Toc516880303)

[3.4.2 Описание работы с «Selenium» 60](#_Toc516880304)

[3.4.3 Основные термины для «Selenium» 61](#_Toc516880305)

[3.4.4 Применение основного функционала «Selenium» 62](#_Toc516880306)

[3.5 Использование «Selenium Grid» 63](#_Toc516880307)

[3.6 Использование docker контейнера 67](#_Toc516880308)

[3.7 Использование обертки «HtmlElements» 68](#_Toc516880309)

[3.8 Конфигурация запуска тестов с помощью библиотеки TestNG 69](#_Toc516880310)

[3.9 Использование библиотеки AllureReport 70](#_Toc516880311)

[3.10 Процесс развертывания 72](#_Toc516880312)

[3.10 Валидация результатов проектирования 73](#_Toc516880313)

[ЗАКЛЮЧЕНИЕ 79](#_Toc516880314)

[БИБЛИОГРАФИЧЕСКЙ СПИСОК 81](#_Toc516880315)

[ПРИЛОЖЕНИЕ А 82](#_Toc516880316)

# ВВЕДЕНИЕ

С развитием технологий понятие качества становится одним из наиболее важных показателей. Сопряжено это с ярко проявляющейся тенденцией: чем выше уровень развития технологий, чем сложнее и серьезнее решаемые задачи, тем важнее вопрос качества.

В качестве заинтересованы как потребители, так и производители программных решений. Потребители хотят пользоваться качественными товарами и услугами для удовлетворения своих потребностей. Производитель заинтересован, т.к. ему необходимо обеспечивать качество.

В настоящее время разработка программных решений происходит в сжатые сроки и при ограниченных средствах. Значимость проблемы контроля качества программного обеспечения обуславливается темпами развития IT-компаний и их заинтересованности в конкурентоспособности продукции. В такой ситуации разработчики могут игнорировать необходимость обеспечения требуемого уровня качества программных решений, тем самым подвергая реализуемый проект рискам.

Качество не может возникнуть самостоятельно, его необходимо обосновать и обеспечить. Поэтому для успешной деятельности разработки программного решений, обеспечивающей получение намеченных результатов требуемого уровня качества и достижения запланированных целей при экономном использовании ресурсов, требуется, проработка ключевых показателей, которые определяют способность работать качественно и производить качественную продукцию. Ключевые показатели характеризуют динамику развития проекта в измеримых единицах, которые используются для планирования проекта на протяжении всего жизненного цикла программного обеспечения.

Показатели непосредственно связаны с найденными в продукте ошибками – дефектами. Ошибки выявляются на этапе тестирования и эксплуатации пользователями, но необходимо учитывать, что обнаружение и устранение дефектов на ранней стадии разработки ПО обходится компании в разы дешевле, чем на более поздних этапах эксплуатации.

Тестирование включает в себя:

* Проверку соответствия требованиям;
* Наблюдение за процессом тестирования;
* Искусственное создание ситуаций, выбранных определенным образом, т.е. составлением тестов.

Тест представляет собой заранее спланированною ситуацию для наиболее характерного типа выявления ошибки. Различные виды тестирования предполагают проверку основных аспектов качества:

* Функциональность;
* Надежность;
* Практичность;
* Эффективность;
* Сопровождаемость;
* Мобильность.

Существуют различные подходы к тестированию по критерию изолированности:

* Блочное представляет собой тестирование одного в изоляции от остальных;
* Интеграционное представляет собой тестирование группы взаимодействующих модулей;
* Системное это тестирование системы в целом.

Тестирование позволяет агрегировать информацию об уже найденных в процессе разработки ошибок, что позволяет понимать текущее состояние продукта. Актуальное поступление сообщение об дефекте менеджерам проектов, разработчикам и аналитикам в значительной мере снижает вероятность повторения ошибок, что соответственно положительно влияет на качестве программного обеспечения, а также позволяет контролировать ход разработки путем своевременного перераспределения ресурсов.

Для отслеживания дефектов существуют баг-трекинговые системы. Их основная задача заключается в учете и контроле ошибок и неисправностей, найденный в программных решениях, запросов пользователей и отслеживанием процесса устранения этих ошибок и выполнения пожеланий пользователей. Основным компонентом подобной системы является база данных, которая включает в себя сведения об обнаруженных дефектах.

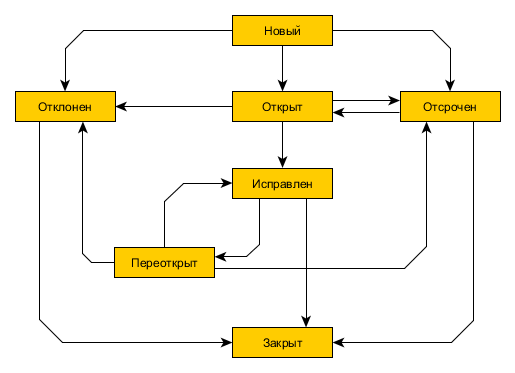


Рисунок 1 - Жизненный цикл дефекта в баг-трекинговой системе[[1]](#footnote-1)

Объектом исследования является процесс разработки программного обеспечения на примере продукта «ZOZO WFM». Предметом исследования является система верификации.

Цель работы:

Спроектировать комплекс мероприятий для контроля качества в рамках процесса разработки ПО с помощью системного подхода.

Задачи работы

1. Рассмотреть основные понятия, касающиеся проблемы исследования: качество в процессе разработки ПО, контроль качества, обеспечение качества.
2. Изучить процесс разработки ПО и выделить основные аспекты, влияющие на качество.
3. Выявить инструмент для осуществления тестирования ПО.
4. Разработать практические рекомендации по усовершенствованию процесса разработки ПО.

# ГЛАВА 1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ КОНТРОЛЯ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 1.1 Описание жизненного цикла разработки программного обеспечения

При рассмотрении жизненного цикла разработки программного обеспечения будут использоваться представление системного инженера [1] разделенное на этапы:

* Разработка концепции;
* Разработки инженерно-технических решений;
* Постразработческая стадия.

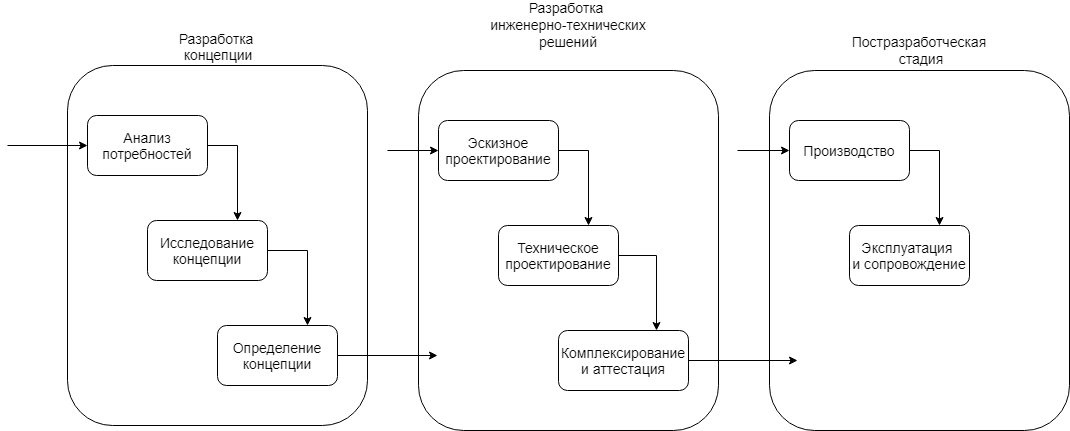


Рисунок 2 - Модель жизненного цикла, предназначенная для системного инженера[[2]](#footnote-2)

Для детализации жизненного цикла чтобы обозначить проблему контроля качества будет использоваться V-диаграмма. Эта модель обеспечивает поддержку в планировании и реализации проекта, и тем самым решает следующие задачи [2]:

* Минимизация рисков;
* Повышение и гарантии качества;
* Уменьшение общей стоимости проекта;
* Повышение качества коммуникации между участниками проекта.

Разработка программного продукта на этапе детализации концепции включает в себя анализ и планирование для выяснения потребности в новой системе, техническую возможность ее воплощения и наилучшую архитектуру для удовлетворения нужд потребителя. Основными целями на этой стадии является:

* Выяснить что есть потребность в новой системе на рынке существует, а также, что имеются технические и экономические возможности ее реализации;
* Исследовать возможность концепции системы;
* Выбрать наиболее подходящую концепцию, определить ее основные характеристики и составить детализированный план работ;
* Разработать новые технологии, которые необходимы для реализации выбранной концепции.

На этапе инженерно-технических решений происходит реализация функций системы исходя из требований спецификации, реализованных на предыдущем этапе, в физическом воплощении. Основные цели этапа:

* Реализация инженерно-технической разработки пилотной версии системы, удовлетворяющий функциональный требованиям;
* Создание системы, которая будет экономичная при производстве и эксплуатации.

Постразработческий этап включает в себя все то, что не вошло в этап разработки, но требует поддержку в виде комплекса мер направленных на этапе эксплуатации. Например, модернизации системы без вывода из эксплуатации. Стадия начинается после валидации системы, т.е. после приемочных испытаний.

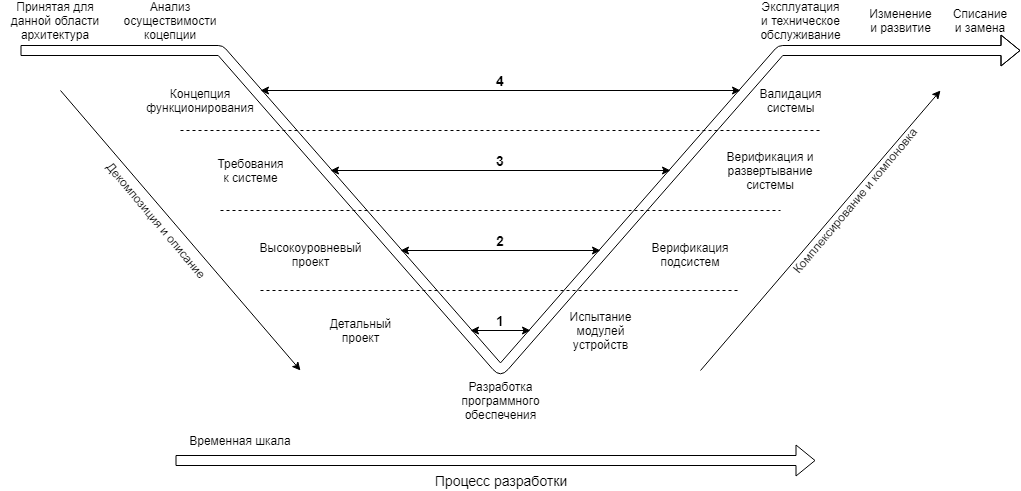


Рисунок 3 - V – диаграмма жизненного цикла разработки программного обеспечения[[3]](#footnote-3)

## 1.2 Основные характеристика качества

Качество – это степень соответствия совокупности присущих характеристик требованиям [3].

Качество программного обеспечения является важным показателем на протяжении всего жизненного цикла. Качество определяет способность удовлетворения потребностей пользователей в рамках проектных ограничений.

Для определения качества выделяют такие характеристики как функциональность, надежность, удобство использования, эффективность, удобство сопровождения, портативность [4].

Функциональность включает в себя функциональную исправность, соответствие стандартам, функциональную совместимость, безопасность и точность системы. Функциональность является основным аспектом качества. Характеристики функциональности:

* Пригодность к использованию, т.е. система должна делать то, что она должна, то, для чего оно предназначена.
* Точность, т.е. система должна выполнять работу с определенной точностью.
* Способность к взаимодействию с другими системами, т.е. способность взаимодействия с операционный окружением и поддерживать определенные стандарты.
* Защищенность,

Надежность включает в себя такие характеристики как:

* Зрелость или завершенность, определяет свойства системы начинаю от начального уровня с без четкого планирования, заканчивая уровнем оптимизации.
* Устойчивость к отказам — это способность системы не реагировать на воздействия.
* Восстановление работоспособности при отказах — это способность системы к самовосстановлению после отказа.

Удобность использования включает в себя такие характеристики как:

* Понятность программы, т.е. пользователь системы должен понять, как воспользоваться ею для достижения своих целей.
* Удобство изучения это способность обучить пользователя системы самостоятельно получать требуемый результат.
* Работоспособность — это возможность пользователю системы управлять поведением системы, системы должна соответственно реагировать.
* Привлекательность это способность привлекать конечного пользователя своей совокупностью характеристик.

Эффективность включает в себя такие характеристики как:

* Временные характеристики: время отклика, скорость работы системы, скорость обработки определённых и т.д.
* Использование ресурсов — это количественная оценка используемых ресурсов для поддержания системы.

Удобство сопровождения включает в себя такие характеристики как:

* Анализируемость кода
* Изменяемость это способность вносить изменения в системы.
* Риск возникновения неожиданных эффектов после внесения изменений, т.е. вероятность реакции на воздействие системы.
* Контролируемость это способность системы быть протестированной.

Портативность включает в себя такие характеристики как:

* Простота установки.
* Одновременная работа с другими системами.
* Способность к заменяемости аналогичных систем.

## 1.3 Основные характеристики контроля качества

Контроль качества одна из основных функций в процессе управления качеством. Контроль позволяет вовремя выявлять ошибки, чтобы затем оперативно исправить их с минимальными затратами. Выполнений функций проведения контроля качества в разработке программного обеспечения берут на себя тестировщики. Для фиксирования определенного уровня качества существуют различные метрики, например: отношение удачно выполненных тестов к числу проводимых тестов. По метрикам можно судить о качестве продукта, о его динамике развития, так же корректировать направление развития.

## 1.4 Обеспечение качества

Обеспечение качества состоит из контроля качества и мероприятий направленных для достижения определённого уровня качества.

Обеспечения качества — это комплекс мероприятий на протяжении всего жизненного цикла разработки программного обеспечения включающий мероприятия для достижения определенного уровня качества. Начиная от этапа выявления потребностей стейкхолдеров, где QA-инженеры валидируют требования к возможности реализации в контексте текущего состояния системы. На основе этого можно принимать решения о целесообразности модификации системы. Для принятия подобного решения требуется проанализировать способность системы к изменениям и спрогнозировать варианты развития качества продукта. Это ведет за собой изменение сценариев тестирования и обновление документации по тестам.

## 1.5 Обзор существующих подходов к контролю качества

Современные компанию используют в зависимости от необходимости два вида тестирования:

* Ручное;
* Автотестирование.

Их соотношение между собой определяет эффективность тестирования.

На ручное тестирование уходит много времени, но в кратковременной перспективе экономит на порядок больше ресурсов команды разработки. Стоимость тестирования напрямую зависит от тестировщика, т.е. решающую роль играет профессионализм человека, а не аппарат автоматизации. Основными плюсами такого тестирования является:

* Предложение по улучшению UI/UX , т.к. пользовательский интерфейс можно тестировать только в ручную;
* Стоимость тестирования. В кратковременной перспективе ручное тестирование экономически целесообразнее, чем средства автоматизации;
* Обратная связь от предполагаемого пользователя системы;
* Исследовательское тестирование;
* Нет необходимости в написании кода для проверка небольших изменений в функционале.

Основными минусами ручного тестирования являются:

* Наличие человеческого фактора;
* Неэффективное регрессионное тестирование;
* Невозможность моделирования нагрузочного тестирования.

Многие программные приложения сегодня написаны как веб-приложения для запуска в интернет-браузере. Эффективность тестирования этих приложений в значительной мере отличается среди компаний и организаций. Автоматизация означает использование программного инструмента для запуска повторяющихся тестов, что для регрессионного тестирования это обеспечивает гибкость.

Автотестирование представляет с собой написание скриптов, имитирующих действия пользователя и их поддержку. Оно играет ключевую позицию в программных решениях, где присутствует большое количество функций. Основными плюсами являются:

* Исключение человеческого фактора, что позволяет беспристрастней взглянуть на общую картинку качества в целом;
* Скорость воспроизведение сценарием на порядки выше чем у человека, что в свою очередь дает большое преимущество в регрессионном тестировании;
* Поддержка теста в виде кода, если затрагиваемый функционал не изменялся, не требуется;

Основными минусами являются:

* Стоимость. Средства автотестирования и обучения их использованию в разы дороже чем обучения ручному тестированию;
* Невозможность проверки UI;
* Существуют ошибки, которые может выделить только человек.

Как и любое высокотехнологическое решение автотестирование позволять достичь значимых результатов скорости тестирования, но при этом необходимы ощутимые вложения в виде ресурсов. Иначе пирамиде тестирования может превратится в «рожок с мороженным», где автоматизация не используется полноценно, рисунок 4.

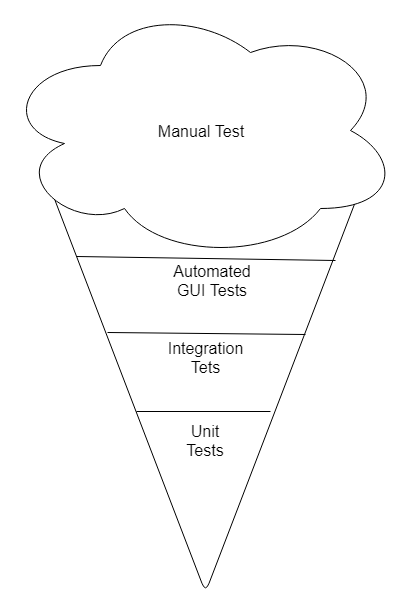


Рисунок 4 - Обратная пирамида тестирования[[4]](#footnote-4)

**ГЛАВА 2 АНАЛИЗ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА «ZOZO WFM»**

## 2.1 Характеристика деятельности системы «ZOZO WFM»

WFM-система представляет собой автоматизацию процесса управления рабочим временем персонала. Основной целью является их является использование повешение производительности за счет:

* Составление оптимального расписания с учетом их квалификации, специализации и пожеланий сотрудников;
* Планирование численности персонала на основании анализа работы за предыдущие периоды времени и т.д.

Эффективность ритейла зависит от того, как он организован, насколько точно спланированы смены сотрудников и рассчитаны возможные пиковые нагрузки. Системы WFM позволяют сделать рабочие процессы прозрачными т.к. используются ключевые параметра такие как: количество сотрудников и степень их загрузки, очередь, точность соблюдения расписания, мониторинг объёма обработанных клиентов. При использовании подобных комплексных систем обрабатываются такие проблемы как:

* Более эффективное использование персонала – решения для распределения рабочей нагрузки, которое обеспечивает прогноз объема работы и более точное планирование, что позволяет оптимально формировать численность сотрудников по сменам.
* Распределять сотрудников по имеющимся профессиональным навыкам – решения для оптимизации укомплектованности штата в течение дня для соответствия текущим потребностям.
* Строгое соблюдение рабочего расписания – решение позволяет получать информацию о текущей эффективности работы персонала, соблюдения ими расписания.
* Уменьшение лишней рабочей силы – решение обеспечивает точное планирование работы сотрудников на основе прогноза ожидаемого объёма работ.
* Снижение сверхурочных расходов – прогноз и составления расписания позволяют осуществлять мониторинг объем сверхурочных работ в реальном времени.

## 2.2 Использование методологии «черный ящик»

В первой итерации определения стейкхолдеров используется подход «черный ящик» для определения границ использующей системы.



Рисунок 5 - Модель «черного ящика» использующей системы[[5]](#footnote-5)

На основании общей концепции ведения бизнеса основная цель компании — это привлечение клиентов для дальнейшей продажи товаров и услуг. На рисунке 3 можно выделить основной входящий поток в виде запросов клиентов. Пожелания клиентов оказывают влияние на процессы внутри «черного ящика», а именно определение возможности изменение настроек для внедрения продукции или же необходимость разработки дополнительных программных решений. Поскольку «ZOZO WFM» является уже готовым набором решений для организации ведения бизнеса, выходным потоком является система в эксплуатации. Остальные политические аспекты, влияющие на «черный ящик», которые могут вносит эффект случайных событий, будут рассмотрены как внешний регулятор.

## 2.3 Определение операционного контекста проектируемой системы

Для определения операционного контекста необходимо выделить основные взаимодействующие между собой системы [ссылка на приложение]. Поскольку границы использующей системы обозначены можно начинать исследовать каким образом входной поток в виде запросов клиентов преобразуется в поток по типу система в эксплуатации.

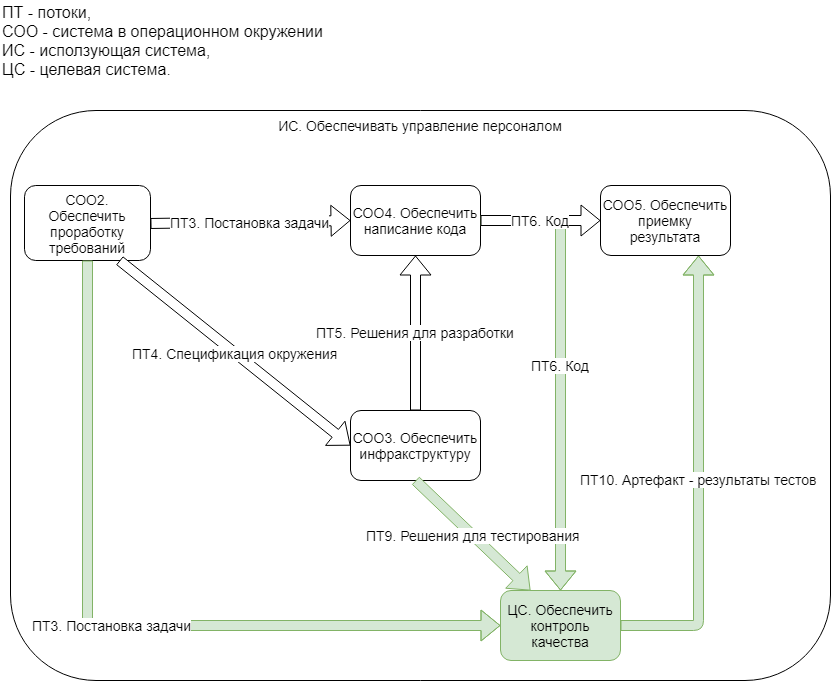


Рисунок 6- Затрагиваемый операционный контекст проектируемой системы[[6]](#footnote-6)

### 2.3.1 Функция «Обеспечить работу с клиентом»

Запросы клиентов преобразуются функцией «Обеспечить работу с клиентом», на этом этапе подключается в работу консалтинг для детализации запроса клиента на изменения системы. В результате имеется первоначальное представление о запросах клиента для выявления нужд и требований заказчика. Консалтинг занимается исследованием следующих сфер деятельности:

* Управленческая;
* Финансовая;
* Инвестиционная;
* Экономическая.

### 2.3.2 Функция «Обеспечить проработку требований»

Функция «Обеспечить проработку требований» в лице аналитиков преобразует требования в артефакт в виде технического задания. Основные действия этой системы включают в себя:

* Получение, формализация и согласование требований клиента;
* Подготовка информации, описание и моделирование бизнес-процесса;
* Анализ эффективности выбранных решений и их оптимизация;
* Составление сравнительно анализа проектной деятельности;
* Презентация работ для клиентов.

Задание включает в себя описание постановки задач для дальнейшей реализации с учетом потребностей и нужд заказчика. В том числе аналитики занимаются разработкой регламентирующей документации.

### 2.3.3 Функция «Обеспечить инфраструктуру»

На основании технических требований создается советующая инфраструктурная среда функцией «Обеспечить инфраструктуру». Инженеры администрирования занимаются развертыванием структуры разработки:

* Серверная часть;
* Удаленные рабочие места
* Настройка процесса развертывания системы
* Автоматизация процесса развертывания
* Настройка баз данных и т.д.

### 2.3.4 Функция «Обеспечить написание кода»

После того как подготовлена среда для разработки и детализированы задачи функция «Обеспечить написание кода» преобразует в артефакт в виде «кода». Команда разработчиков делится на две основные части:

* Разработчики front-end
* Разработчика back-end.

Разработчики front-end берут на себя реализацию внешнего вида программного обеспечения, т.е. его клиентской части. Разработчики back-end занимаются обратной стороной ресурса, а именно, тем что не видно пользователю. Они реализуют механизмы работы программного обеспечения с базами данных и серверной частью приложения.

### 2.3.5 Функция «Обеспечить приемку результата»

Функция «Обеспечить приемку результата» валидирует систему с тем, что ожидали получить и получилось в итоге. Эта система в операционном окружении обеспечивает контрактацию множества специалистов для доведения продукта до состояния готового для демонстрации пользователю. Происходит валидация системы в целом на соответствие требованиям и первоначальной концепции. Основные процессы, охватывающие функцию «Обеспечить приемку результата»:

* Валидация системы;
* Верификация требований;
* Анализ результатов тестирования;
* Согласование изменений в работу программного обеспечения;
* Доработки на основании анализа результатов тестирования.

### 2.3.6 Функция «Обеспечить контроль качества»

Целевая система «Обеспечить контроль качества» взаимодействует с техническим заданием и преобразует его в тест-планы, для того чтобы спланировать необходимые специально выбранные проверки.

Так же используются решения для тестирования, а именно, помощь инженеров администрирования в настройке среды для тестирования, создание тестовых сред и организацией непрерывного развертывания продукта в тестовой среде.

Поток в виде кода это непосредственно сам продукт. Тестируоемое приложение подвергается регламентированным проверкам на основании принятых методик тестирования.

В итоге необходимо оценить качество текущего состояния продукта по средством проведения тестирования.

## 2.4 Определение стейкхолдеров системы «ZOZO WFM»

На основании того, что границы проектируемой системы определены, а также выяснены основные системы в операционном окружении можно выделить стейкхолдеров [5]. Функции систем, представленные во время операционного проектирования, дают возможность определить их исполнителей, а значит и стейкхолдеров.

Таблица 1. Стейкхолдеры системы «ZOZO WFM»

|  |  |  |
| --- | --- | --- |
| ИД | Стейкхолдер | Исполнитель |
| СХ1 | Заказчик системы | Владелец бизнеса |
| СХ2 | Заказчик и инвестор проекта | Директор ИТ-компании |
| СХ3 | Пользователь системы | Пользователь системы |
| СХ4 | Аналитик | Представитель отдела аналитики |
| СХ5 | Инженер администрирования | Представитель отдела инфраструктурных решений |
| СХ6 | Разработчик программного обеспечения | Представитель отдела разработки ПО. |

На основании бизнес-ролей определяются потребности и проблемы, которые их вызывают.

Таблица 2. Потребности стейкхолдеров системы

|  |  |  |  |
| --- | --- | --- | --- |
| ИД | Стейкхолдер | Потребность | Проблема |
| П1 | Заказчик системы ZOZO | Контролировать персонал | Распределение ресурсов |
| П2 | Инвестор проекта | Зарабатывать в нище систем WFM | – |
| П3 | Пользователь системы | Контролировать свое расписание | Распределение рабочего времени |
| П4 | Аналитик | Верифицировать ТЗ | Верификацией поставленных задач |
| П5 | Инженер администрирования | Обеспечивать непрерывное развертывание системы (про DevOps CI/CD) | Нестабильность сборки |
| 1 | 2 | 3 | 4 |
| П6 | Разработчик программного обеспечения | Понимать место как воспроизведения ошибки | Допускаются ошибки |

### 2.4.1 Заказчик системы «ZOZO WFM»

Я, как заказчик системы, хочу увеличить конверсии с продаж, потому что нет объективной картины как работает персонал.

Я, как заказчик системы, хочу экономить на фонде оплаты труда, потому что считаю распределение персонала с учетом специализации и уровня профессионализма в течение рабочей смены не целесообразным.

Я, как заказчик системы, хочу контролировать персонал, потому что это исключает человеческий фактор при составлении расписания.

Я, как заказчик системы, считаю, что системы должна количественно прогнозировать конверсию при определенном штате, чтобы появилось объективная картина того, как работает персонал.

Я, как заказчик системы, считаю, что система должна проводить балансировку часов сотрудников внутри рабочего дня, чтобы распределение персонала зависимости от их специализации и уровня профессионализма было эффективным.

Я, как заказчик системы, считаю, что система должна автоматически составлять расписание, что исключить человеческий фактор.

### 2.4.2 Инвестор проекта «ZOZO WFM»

Я, как инвестор проекта «ZOZO WFM», хочу зарабатывать в нище систем WFM.

Я, как инвестор проекта «ZOZO WFM», считаю, что система должна предоставлять сведения о текущем положении дел.

### 2.4.3 Пользователь системы «ZOZO WFM»

Я, как пользователь системы, хочу, чтобы мои пожелания были учтены при составлении расписания, чтобы планировать личного времени.

Я, как пользователь системы, хочу контролировать свое расписание с учетом планирования работы организации, чтобы планировать личное время.

Я, как пользователь системы, считаю, что система должна предоставлять возможность оставлять личные пожелания при составлении рабочего расписания.

Я, как пользователь системы, считаю, что система должна предоставлять заблаговременные сведения о изменениях в расписании.

### 2.4.4 Аналитик

Я, как аналитик, хочу верифицировать техническое задание, чтобы уменьшить количество ошибок в постановке задач.

Я, как аналитик, считаю, что система должна обеспечивать стандартизацию составления тест-планов, чтобы верифицировать техническое задание.

### 2.4.5 Инженер администрирования

Я, как инженер администрирования, хочу обеспечивать непрерывное развертывание системы, чтобы уменьшить время по администрированию.

Я, как инженер администрирования, хочу понимать необходимый набор инструментов для тестирования, чтобы понимать взаимодействие между информационной инфраструктурой.

Я, как инженер администрирования, считаю, что система должна работать согласно регламентной документации, чтобы понимать набор инструментов для тестирования.

Я, как инженер администрирования, считаю, что система должна с оптимальным набором приложений, чтобы обеспечить непрерывное развертывание системы.

### 2.4.6 Разработчик программного обеспечения

Я, как разработчик программного обеспечения, хочу понимать как локализовать ошибку, чтобы исправить несоответствующий требованиям функционал.

Я, как разработчик программного обеспечения, хочу понимать какими инструментами пользуется тестирование, чтобы организовывать удобные интерфейсы для тестирования.

Я, как разработчик программного обеспечения, считаю, что система должна иметь стандартизованный шаблон описания ошибок, чтобы понимать, как локализовать ошибку.

Я, как разработчик программного обеспечения, считаю, что система должна описывать подходы к тестированию, чтобы понимать какими инструментами пользуется тестирование.

## 2.5 Моделе-ориентированные бизнес-анализ

### 2.5.1 Функциональный бизнес-контекст

В процессе проектирования системы контроля качества принимают участие такие специалисты как:

* Инженеры тестирования;
* Инженеры администрирования;
* Аналитики;
* Разработчики программного обеспечения.

Мнение тестировщиков обеспечивает взгляд человека, непосредственно проводящего тестирование и контроль качества. Собирается опыт тестирования, каким образом лучше описывать ошибки, как легче всего выявить характерные критичные ситуации, как спланировать объем работ по тестированию. На основании этого разрабатывается подход к тестированию.

Инженеры администрирования вносят свои коррективы каким образом лучше развернуть приложение для выявление узких мест.

Аналитики дает виденье со стороны постановки задач, что помогает спланировать тестирование и подготовить структуру тест-планов.

Разработчики предлагают решения на тему целесообразного подхода к тестируемому программному обеспечению с учетом алгоритмов при которых было разработано программное обеспечение.

В итоге результата генерации необходимо фреймворк контроля качества, основанный на мнениях всех участников разработки, рисунок А1.

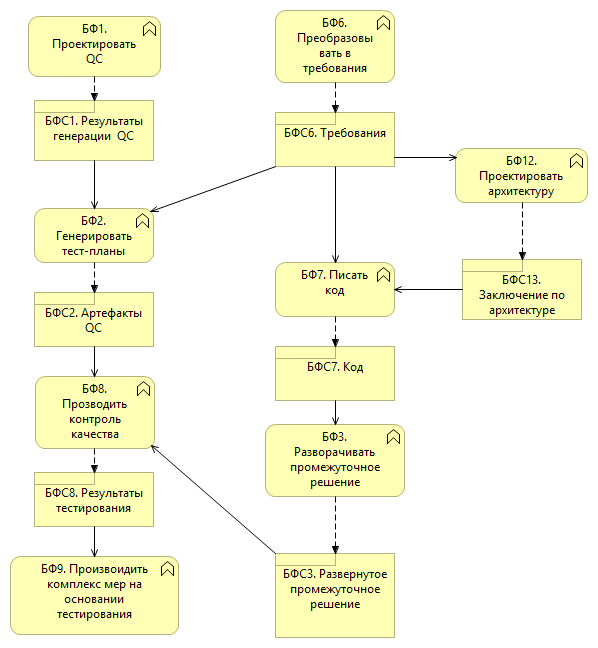


Рисунок 7 - Часть функциональный бизнес-контекста[[7]](#footnote-7)

При генерации тест-планов учтен опыт участников разработки. Тест-план — это документ, в котором излагается весь объем по тестированию, и он включает в себя:

* Описание объекта тестирования;
* Стратегия;
* График работ;
* Условия начала и завершения тестирования
* Требуемое тестовое окружение и т.д.

При автоматизации тестирования на этом этапе идет разработки кода для имитации действий пользователя и нагрузки системы, с учетом установленных ограничений от других участников разработки.

Во время непосредственного тестирования используется ранее сгенерированный план работ, заполняются подготовленные метрики на основании которых появляются результаты тестирования. При автоматизации запускаются тестовые среды, активируются скрипты, которые имитируют пользователей. По окончанию работы скрипта создается авто генерируемый отчет.

### 2.5.2 Структурная модель

При построении бизнес-структурной модели были выделены основные интерфейсы взаимодействия между участниками процесса разработки, рисунок А 2. Все участники команды посещают ежедневые собрания и рассказывают о текущем статусу задач для понимания общего состояния процесса.

Аналитики после взаимодействия с заказчиком детализируют и описывают задачи в «Confluence». Это открытая система хранения информацией для корпоративных решений. Через описание задачи в базе знаний разработчики понимают, что не посредство необходимо реализовать в виде кода. Инженеры тестирования в той же базе знаний описывают тест-планы и уточняют корректность формулировок у аналитиков. Архитектор системы предоставляют консультации для разработчиков. В большинстве интерфейсов взаимодействия остается устное общение между члена команды.

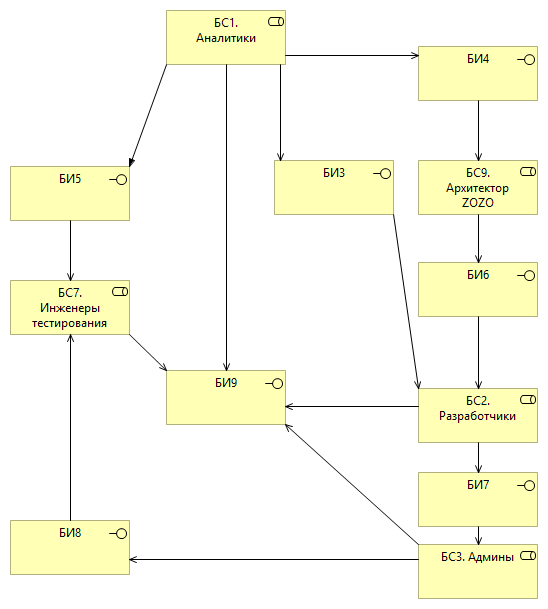


Рисунок 8 - Часть структурной модели[[8]](#footnote-8)

Инженеры администрирования обеспечивают доставку промежуточного решения на сервера для тестирования. На основании результатов тестирования команда проводит дополнительные итерации по разработке по необходимости.

### 2.5.3 Модель стейкхолдеров

Для уточнения стейкхолдеров рисунок А 3, выявленных ранее методом «черного ящика», использовалась матрица для определения стейкхолдеров. На пересечении бизнес-функции и бизнес-структуры расположены стейкхолдеры, рисунок 8.

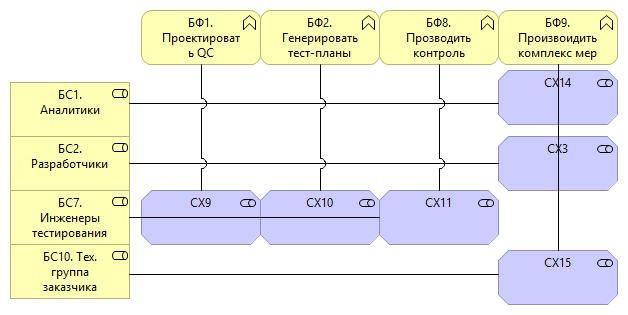


Рисунок 9 - Матрица определения стейкхолдеров[[9]](#footnote-9)

### 2.5.4 Определение целей стейкхолдеров

Для уточнения бизнес-целей, использовалась матрица определения бизнес-целей, рисунок А 4. На пересечении бизнес-интерфейса и бизнес-функциональной связи расположена бизнес-цель, рисунок 10.

После определения бизнес-целей с помощью матрицы, необходимо определить взаимосвязь целей между собой. Это требуется для того чтобы можно было ранжировать главные цели от второстепенных, рисунок А 5.

На рисунке 11 представлены основные цели системы. При проектировании системы контроля качества будет учитывать цель «Описать фреймворк тестирования».

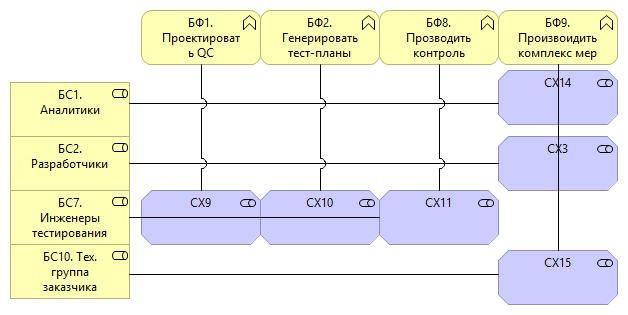


Рисунок 10 - Часть матрицы определения бизнес-целей[[10]](#footnote-10)

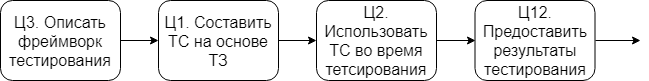


Рисунок 11 - Схема взаимосвязей между бизнес-целями[[11]](#footnote-11)

## 2.6 Моделе-ориентированная работа со стейкхолдерами

### 2.6.1 Функциональная модель системы контроля качества в бизнес-контексте

Первостепенной функцией системы контроля качества является выполнять тестирование и заполнять тест-план. После заполнения тест-плана происходит заполнение метрик для оценки качества. При автоматизации тестирования описывается сценарий воспроизведения в виде скриптов. С помощью функций непрерывной интеграции скрипты отрабатывают на тестовом стенде и генерируется автоотчет. В итоге независимо от режима работы получается отчет для дальнейшей передачи команде разработки.

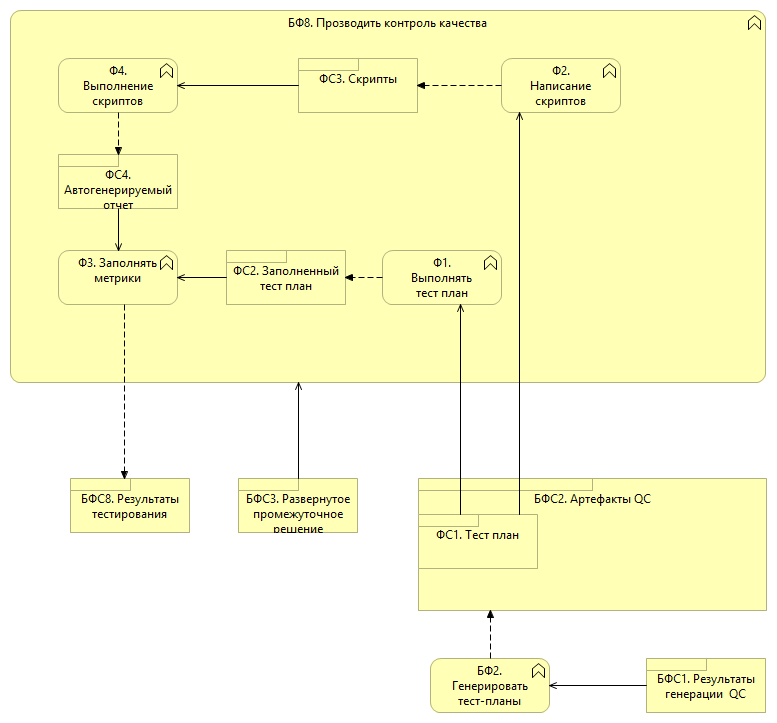


Рисунок 12 - Функциональная модель системы контроля качества[[12]](#footnote-12)

Таблица 3. Функции системы контроля качества

|  |  |  |
| --- | --- | --- |
| ИД | Функция | Описание |
| Ф1 | Выполнять тест-план | Включает в себя проведение тестирования с использованием тест-плана |
| Ф2 | Написание скриптов | Включает в себя написание скриптов, разработку прикладных решений для тестирования, подготовка тестового окружения |
| 1 | 2 | 3 |
| Ф3 | Заполнять метрики | Заполнение метрик на основе которых определяется показатель качества программного обеспечения |
| Ф4 | Выполнение скриптов | Выполнение скриптов программными комплексами на серверах в тестовом окружении |

Таблица 4. Функциональные связи системы контроля качества

|  |  |  |
| --- | --- | --- |
| ИД | Функциональная связь | Описание |
| ФС1 | Тест-план | Набор тестов в рамках тестируемого модуля |
| ФС2 | Заполненный тест-план | Артефакт наполненный данными |
| ФС3 | Скрипты | Код имитирующий действия пользователей |
| ФС4 | Автогенерируемый отчет | Набор метрик и информации о состоянии тестов |

### 2.6.2 Определение требований стейкхолдеров

На основании функционально смоделированной модели системы контроля качества необходимо произвести уточнения требований стейкхолдеров. Используется матрица определения требований представленная на рисунке 13, где на пересечении функциональной связи и стейкхолдера выделены их требования, таблица 5.

Таблица 5. Определение требований стейкхолдеров

|  |  |  |  |
| --- | --- | --- | --- |
| ИД | Стейкхолдер | ИД | Требование |
| СХ1 | Аналитики | ТСХ1 | Понятный тест-план с однозначными формулировками |
| 1 | 2 | 3 | 4 |
| СХ11 | Аналитики | ТСХ7 | Понятный отчет и возможность доступа к нему |
| СХ2 | Разработчики | ТСХ8 | Определенный набор технологий |
| СХ9 | Исполнители тест-планов | ТСХ2 | Однозначное описание тест-плана |
| СХ10 | Инженер тестирования | ТСХ3 | Стандартизация составление тест-плана |
| СХ11 | Инженер автотестирования | ТСХ4 | Стандартизация составление тест-плана для автотестов |
| СХ11 | Инженер автотестирования | ТСХ5 | Определенный набор технологий, стандартизация набора имитируемых действий |
| СХ11 | Инженер автотестирования | ТСХ6 | Стандартизировать методы генерации отчета |

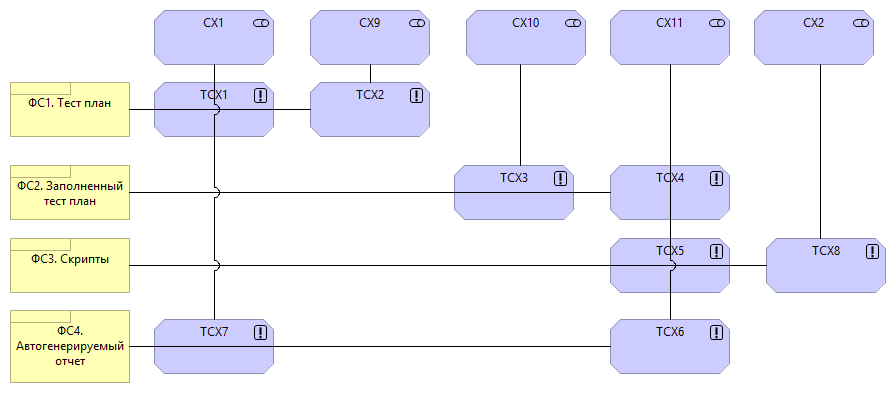


Рисунок 13 - Матрица определения требований стейкхолдеров[[13]](#footnote-13)

### 2.6.3 Сопоставление требований стейкхолдеров

На основании матрицы определение стейкхолдеров были выделены их требования. На рисунке Х представление сопоставление требований стейкхолдеров с их потребностями и ограничениями. Основными ограничениями является набор используемых технологий. При учете потребностей выясняется, что

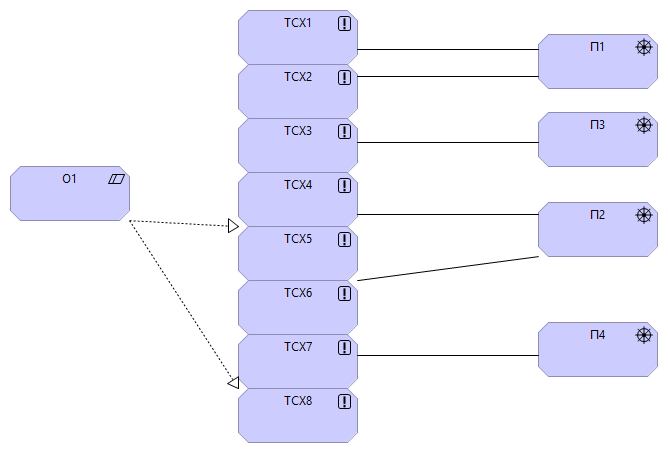


Рисунок 14 - Сопоставление требований стейкхолдеров[[14]](#footnote-14)

Таблица 6. Сопоставление ограничения с требованиями

|  |  |  |  |
| --- | --- | --- | --- |
| ИД | Ограничение | ИД | Требование стейкхолдера |
| О1 | Ограничение на технологии  Back-end Spring Java 8  Front-end Aurelia  Пользовательский набор действий | ТСХ5 | Определенный набор технологий, стандартизация набора имитируемых действий |
| О1 | Ограничение на технологии  Back-end Spring Java 8  Front-end Aurelia | ТСХ8 | Определенный набор технологий |

Таблица 7. Сопоставление потребностей с требованиями

|  |  |  |  |
| --- | --- | --- | --- |
| ИД | Потребность |  | Требование стейкхолдера |
| П1 | Понимать как читать тесты | ТСХ1 | Нужен набор, регламентирующий действий для работы с тестами |
| П1 | Понимать как читать тесты | ТСХ2 | Нужна описанная модель процесса тестирования |
| П2 | Понимать как производить контроль | ТСХ4 | Требования к визуализации, как сообщать об ошибке |
| П2 | Понимать как производить контроль | ТСХ6 | Требование к проведению тестирования, стандартная модели верификации |
| П3 | Разбираться в большом объеме тестов | ТСХ3 | Методика как хранить описание тестов |
| П4 | Понимать как воспроизвести ошибку, как локализуется ошибка | ТСХ7 | Набор требований к описанию тестового случая |

## 2.7 Выбор инструмента тестирования

### 2.7.1 Детализация показателей

Главная цель — повышение скорости и качества тестирования за счет средств автоматизации.

Задача анализа – выявить наиболее оптимальные решения для удовлетворения требований стейкхолдеров.

Вид тестирования - функциональное. Максимальная ценность для стейкхолдеров достигается за счет критериев:

* Минимальные системные требования – объем памяти, загрузка процессора;
* Удобства написания скриптов – возможность быстрого создания кода;
* Минимальные инвестиции – низкая себестоимость и стоимость эксплуатации;
* Эффективность – высокая степень распознавания элементов управления;
* Безопасность – возможности для восстановления сценариев;
* Универсальность – типы тестируемых приложений, виды воспроизводимых тестов, поддерживаемые технологии;
* Простота использования – качество сопровождения, требуемый уровень профессиональных навыков, интеграция с системой разработки и т.д.

Для выделения критериев, имеющих наибольшую ценность, была составлена матрица балансировки весов, представленная на рисунке 15.

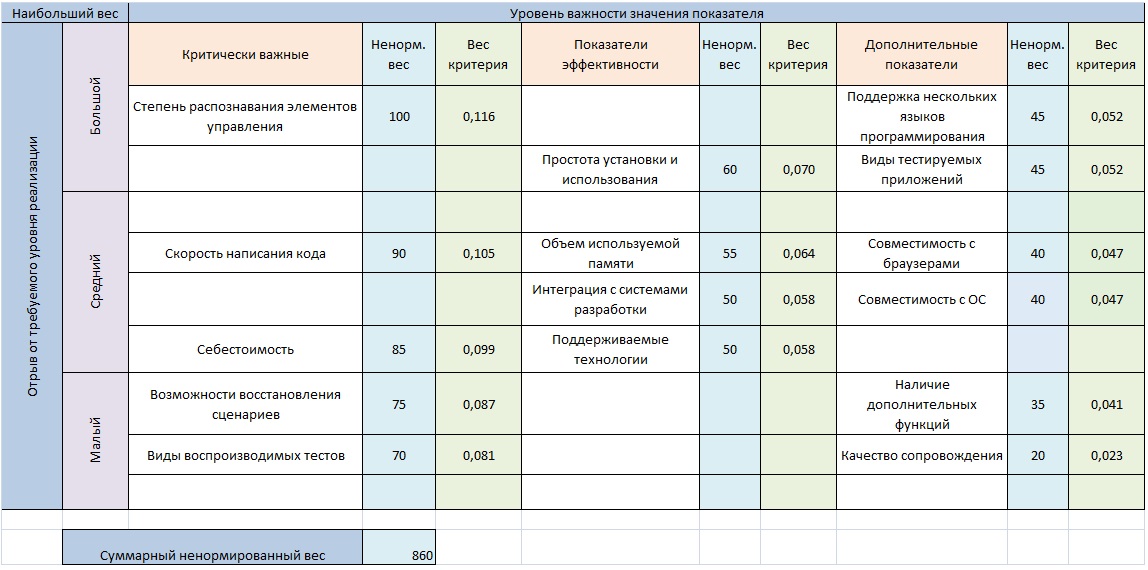


Рисунок 15 - Матрица балансировки весов[[15]](#footnote-15)

К наиболее значимым показателям относятся: степень распознавания элементов управления, скорость написания кода, себестоимость, возможности восстановления сценариев, виды воспроизводимых тестов. Требования к уровню профессиональной подготовки, объему ОЗУ и интеграции представляют собой показатели эффективности. Как дополнительные были выделены показатели: поддержка языков программирования, совместимость с ОС и браузерами, виды тестируемых приложений, наличие дополнительных функций и качество сопровождения.

Экспертным путем для каждого показателя был выставлен балл по шкале от 0 до 100 и получено значение нормализованного веса.

Функции перехода к единой шкале оценки представлены на рисунке 16,17, 18.

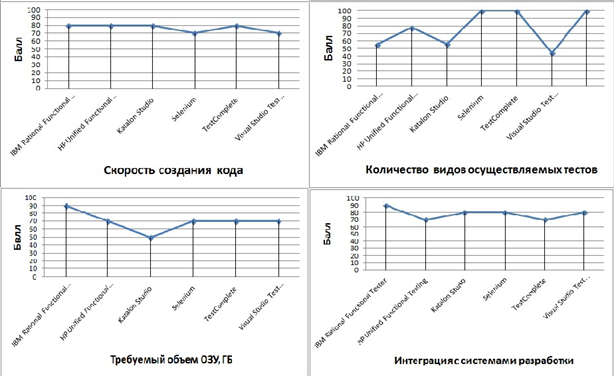


Рисунок 16 – Группа функций перехода[[16]](#footnote-16)

Функции перехода совмещают шкалу измерения с диапазоном оценки, при этом шкала измерений индивидуальна для каждой альтернативы, а диапазон выбирается один на весь проект.

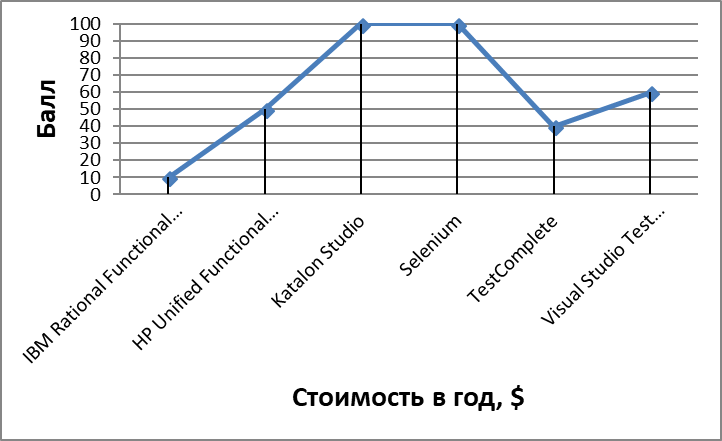


Рисунок 17 - Стоимость лицензии в год[[17]](#footnote-17)

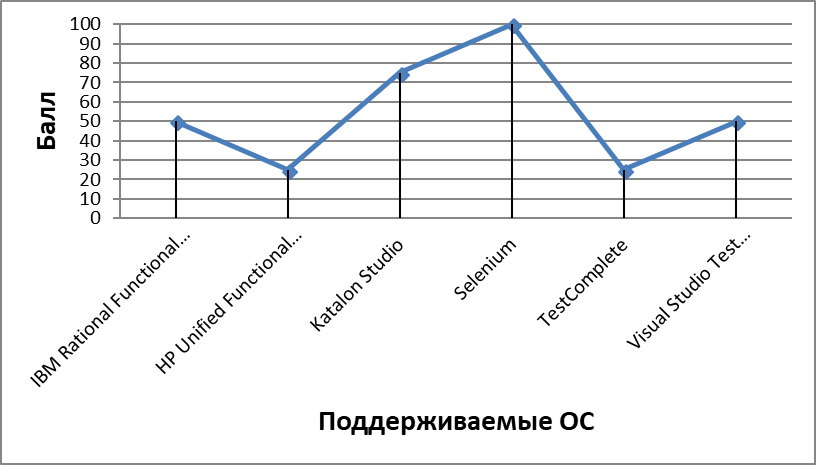


Рисунок 18 – Поддерживаемые операционные системы[[18]](#footnote-18)

### 2.7.2 Описание и оценки альтернатив

В качестве рассматриваемых альтернатив были выбраны следующие программные продукты:

* IBM Rational Functional Tester;
* HP Unified Functional Testing;
* Katalon Studio;
* Selenium;
* TestComplete;
* Visual Studio Test Professional 2017.

IBM Rational Functional Tester – инструмент, обеспечивающий возможность осуществления управляемого данными тестирования функциональности, регрессионного тестирования. Осуществляет тестирование широкого спектра приложений, написанных на различных языках программирования, таких как: .Net, Java, SAP, Flex и Ajax. RFT использует Visual Basic .Net и Java в качестве языков сценариев. Уникальная функция Storyboard testing позволяет записать все связанные с автоматическим тестированием действия пользователей и визуализировать их в виде скриншотов.

HP Unified Functional Testing – один из наиболее популярных коммерческих инструментов для автоматизации функционального тестирования. Включает полный набор функций для тестирования API, веб-сервисов, а также для тестирования графического интерфейса десктопных, мобильных и веб-приложений на всех существующих платформах. Предусмотрена расширенная функция распознавания объектов на основе изображений, многоразовые тестовые компоненты и документация по автоматическому тестированию.

Katalon Studio — это эффективный инструмент для автоматизации процесса тестирования веб-приложений, мобильных приложений и веб-сервисов, созданный на основе открытых фреймворков Selenium и Appium. В отличие от Selenium не требует от пользователей продвинутых навыков программирования и может использоваться даже новичками.

Selenium – один из самых популярных фреймворков с открытым исходным кодом. Является родоначальником таких инструментов для автоматизации тестирования как Katalon Studio, Watir, Protractor и Robot Framework. Отличается поддержкой большого количества браузеров и языков программирования.

TestComplete - инструмент для тестирования десктопных, мобильных и веб-приложений. Поддерживает различные языки сценариев, такие как: JavaScript, VBScript, Python и C ++ Script. Поддерживает тестирование с использованием ключевых слов и управляемое данными тестирование. Имеется функция записи и воспроизведения процесса тестирования. TestComplete осуществляет автоматическое обнаружение и обновление объектов пользовательского интерфейса. Интегрируется с Jenkins в течение CI-процесса.

Visual Studio Test Professional 2017 – интегрированный набор инструментов тестирования для всех этапов рабочего процесса планирования, тестирования и отслеживания, который обеспечивает совместную работу инженеров-испытателей и разработчиков в контексте выполняемых ими задач.

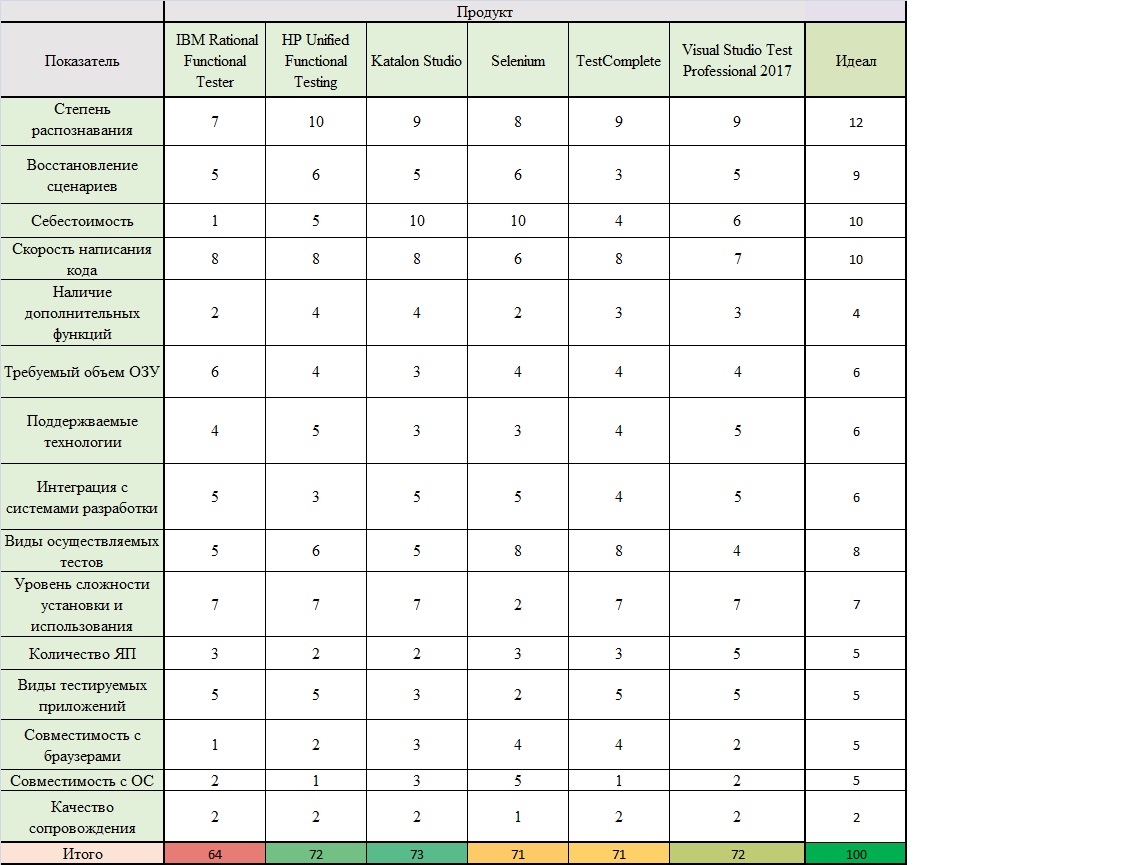


Рисунок 19 - Оценка альтернатив[[19]](#footnote-19)

По данным анализа наибольший балл получили продукты: Katalon Studio (73/100), HP Unified Functional Testing (72/100), Visual Studio Test Professional 2017 (72/100). Следом (71/100) расположились Selenium и TestComplete. На последнем месте - IBM Rational Func-tional Tester (65/100).

Итоговый балл складывается из суммы произведений экспертных оценок по каждому из критериев для каждой альтернативы на нормированный вес показателя из матрицы балансировки весов.

Согласно матрице балансировки весов, критически важным показателем для инструмента автоматизации тестирования является степень распознавания элементов управления. Продукт HP Unified Functional Testing отличается высокой экспертной оценкой по данному критерию. В таблице 8 продукты экспертно сравниваются по степени распознавания элементов с выставлением соответствующих уровню баллов. Система распознавания объектов HP UFT Insight, Smart Object Recognition, image-based object recognition позволяют осуществ-лять тестирование с высокой точностью.

Таблица 8. Степень распознавания элементов управления

|  |  |  |
| --- | --- | --- |
| Продукт | Степень распознавания элементов | Балл |
| IBM Rational Functional Tester | программный интерфейс поиска (find API), технология ScriptAssure | 60 |
| HP Unified Functional Testing | система распознавания объектов HP UFT Insight, Smart Object Recognition, image-based object recognition | 90 |
| Katalon Studio | QTP-like UI/Object распознавание, Smart object capture | 80 |
| Selenium | Локаторы css, link, xpath, dom, id, name, ui | 70 |
| TestComplete | высокое распознавание GUI | 80 |
| Visual Studio Test Professional 2017 | QTP-like UI/Object распознавание | 80 |

Минимальный балл HP Unified Functional Testing получил за совместимость с операционными системами, таблица 9, так как наравне с TestComplete может использоваться только на Windows. Selenium, напротив, имеет максимальную оценку: количество поддер-живаемых им ОС (Mac OS, Microsoft Windows, Linux, Solaris) было взято в качестве идеаль-ного значения для данного показателя. При этом, даже если альтернативное решение будет поддерживать большее число операционных систем, оно не будет иметь дополнительной ценности и получит оценку 100.

Таблица 9. Совместимость продуктов с операционными системами

|  |  |  |
| --- | --- | --- |
| Продукт | Совместимость с ОС | Балл |
| IBM Rational Functional Tester | Linux, Windows | 50 |
| HP Unified Functional Testing | Windows | 25 |
| Katalon Studio | Windows, Mac, Linux | 75 |
| Selenium | Mac OS, Microsoft Windows, Linux, Solaris | 100 |
| TestComplete | Windows | 25 |
| Visual Studio Test Professional 2017 | Windows, Mac | 50 |
| Идеал | Mac OS, Microsoft Windows, Linux, Solaris | 100 |

Для визуализации рисунка 19 на рисунке 20 приведена диаграмма значений по компонентам. Используя данную диаграмму, можно определить вклад каждого показателя в суммарный балл решения.

Рисунок 20 - Диаграмма значений по компонентам[[20]](#footnote-20)

Наибольший вклад в суммарную оценку продукта-лидера Katalon Studio внесли показатели: себестоимость (10/10), степень распознавания элементов управления (9/12), скорость написания кода (8/10) и простота установки и использования (7/7). Наименее значимыми оказались показатели качества сопровождения и количества поддерживаемых языков программирования.

Еще один бесплатный продукт Selenium получил 10 баллов за себестоимость и по 8 за ассортимент осуществляемых тестов и степень распознавания элементов. Слабыми местами данного решения являются качество сопровождения (1/2) и количество предлагаемых языков программирования (2/5).

Расположившийся на 2 месте продукт HP Unified Functional Testing получил максимальный балл по степени распознавания элементов управления (10/10), также он представляет возможности для быстрого написания кода (8/10) и не требует углубленных знаний в области программирования (7/7).

### 2.7.3 Анализ эффективности, рисов, перспектив развития

В таблице 10 приведены оценочные значения себестоимости, функциональности, рисков и перспектив развития рассматриваемых продуктов.

Таблица 10. Оценка себестоимости, функциональности, рисков и перспектив

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Продукт | Себестоимость,$ | Функциональность | Риски | Перспективы развития |
| IBM Rational Functional Tester | 13100 | 64 | 40 | 60 |
| HP Unified Functional Testing | 3200 | 72 | 20 | 60 |
| Katalon Studio | 0 | 73 | 50 | 80 |
| Selenium | 0 | 71 | 30 | 70 |
| TestComplete | 8933 | 71 | 20 | 60 |
| Visual Studio Test Professional 2017 | 2169 | 72 | 20 | 60 |

Риски и перспективы развития оцениваются методом экспертных оценок.

Для оценки рисков внедрения использовались такие критерии как размер и срок пребывания на рынке компании-разработчика, наличие официальной технической поддержки, а также отзывы пользователей.

Таблица 11. Длительность пребывания на рынке и наличие официальной технической поддержки

|  |  |  |  |
| --- | --- | --- | --- |
| Продукт | Компания - разработчик | Наличие техподдержки | Год первого выпуска |
| Rational Functional Tester | IBM | + | 2002 |
| Unified Functional Testing | HP | + | 1998 |
| Katalon Studio | Katalon LLC | - | 2016 |
| Selenium | ThoughtWorks | - | 2004 |
| TestComplete | SmartBear | + | 1999 |
| Visual Studio | Microsoft | + | 1997 |

Целостное представление о системе складывается в результате рассмотрения в совокупности оценок по нескольким критериям. На рисунке 19 представлена диаграмма разброса ценностей, объединяющая перечисленные в таблице 11 критерии. По оси Х откладываются затраты на приобретение лицензии, по y - суммарный балл, определяющий функциональность продукта. Цветом обозначены риски внедрения решения: красный – уровень риска высокий, зеленый – риск использования продукта минимален.

Оценить перспективы развития позволяет размер объектов на диаграмме.

Рисунок 21 - Диаграмма разброса ценностей[[21]](#footnote-21)

Наиболее выгодные для внедрения с точи зрения соотношения себестоимость-функциональность решения расположены на диаграмме в левом верхнем углу. В благоприятную область попадают Katalon Studio, Selenium, Unified Functional Testing, Visual Studio Test Professional 2017.

Лидер по сумме баллов Katalon Studio имеет наибольшие риски внедрения из-за небольшого срока использования на рынке и отсутствия официальной технической поддержки.

Продукты крупных компаний Microsoft, SmartBear и HP проверены временем и вызывают большее доверие у пользователей.

Katalon Studio – новый продукт на рынке, выделяющийся среди других альтернатив своими перспективами развития. Katalon с каждым днем приобретает все большую популярность на рынке средств автоматизации тестирования. Отсутствие вложений и хороший функционал дают возможность для дальнейшего активного развития продукта.

Также хорошие перспективы развития имеет Selenium, пользующийся популярностью у тестировщиков разных стран.

### 2.7.4 Рекомендации по выбору инструмента

В ходе выполнения многокритериального анализа инструментов автоматизации тестирования из общего списка альтернатив выделены следующие решения

Katalon Studio – современный инструмент автоматизации тестирования. Главные преимущества: свободный доступ, низкие требования к уровню программирования, скорость создания скриптов, поддержка основных браузеров и операционных систем, высокая степень распознавания элементов управления, наличие расширенных функций аналитики Katalon Analytics. Недостатки: отсутствие официальной технической поддержки, недостаточная изученность.

Selenium – не теряющий в настоящее время популярности фреймворк с открытым кодом. Проигрыш по общей сумме баллов компенсируется отсутствием платы за лицензию. Преимущества: свободный доступ, совместимость с большинством браузеров и операционных систем, поддержка большого числа языков программирования, широкий ассортимент осуществляемых тестов. Недостатки: требуются навыки программирования, слабое техническое сопровождение.

Преимущества в свободном доступе также предоставляет общение с большим количеством коммитеров – разработчиков открытого исходного кода, которые разрабатывают продукт сами для себя. При коллективном решении технологических проблем возможна положительная синергия в рассматриваем вопросе, т.к. любая технология требует поддержки, то желательно получать опыт практикующих профессионалов. Стоит учесть, что сообществе российского тестирования достаточное количество разработчиков-коммитеров Selenium.

# ГЛАВА 3 ПРОЕКТИРОВАНИЕ СИСТЕМЫ ВЕРИФИКАЦИИ

## 3.1 Разработка подхода к описанию тест-плана

Разработан шаблон оформления тест-плана. Название тест-плана характеризует основную идею набора тестов. В виде тестирования указывается:

* Функциональное;
  + Компонентное;
  + Интеграционное;
  + Системное;
* Нефункциональное;
  + Тестирование производительности;
  + Тестирование установки;
  + Тестирование удобства пользования;
  + Тестирование на отказ и восстановление;
  + Конфигурационное тестирование;
* Связанное с изменениями
  + Дымовое;
  + Регрессионное;
  + Тестирование сборки;
  + Проверка согласованности;

План разделен на разделы:

* Предварительная настройка, включает в себя комплекс этапов для создание тестового окружения;
* Тесты, представляет собой алгоритм действий, характеризуется атомарностью проверки функции в зависимости от типа тестирования;
* Откат изменений для последующего циклического тестирования, предотвращает появление не консистентных данных и влияет на тестовое окружение.

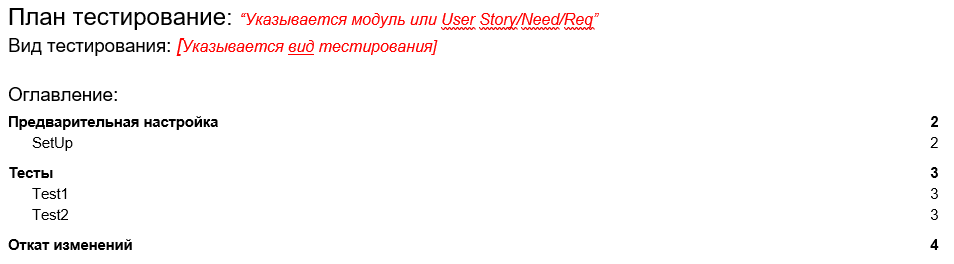


Рисунок 22 - Оглавление тест-плана[[22]](#footnote-22)

Для заполнения теста используется таблица с характерными полями с установленным содержанием:

* На рисунке 22 в поле имя указано «SetUp», отображено имя метода предварительной настройки;
* Цель этапа – указывается на основании того какой функционал хотим проверить, в работе чего на основании этого теста мы можем убедится;
* Условия тестирования – указывается среда тестирования, параметры без которых запуск не возможен. Указывается группа тестов при наличии, например: «Настройка – SetUp, Группа: А1-1, Браузер: Google Chrome, Версия 62.0.3202.94, (64 бит)»
* Действие – содержит алгоритм шагов в виде пронумерованного списка для выполнения теста;
* Ожидаемый результат – указывается описание ожидаемого результата после выполнения всех шагов (или отдельно для каждого шага, если это необходимо проверки);
* Фактический результат/Комментарий – указывается фактический результат выполнения теста; совпал с ожидаемым или нет. При несоответствии указывается что не сошлось;
* Приложение – указывается ссылки на скриншоты и записи, возможно добавление дополнительного поля «Предложение»;
* Предложение – в нем описывается предложения по улучшению или оптимизации функционала.

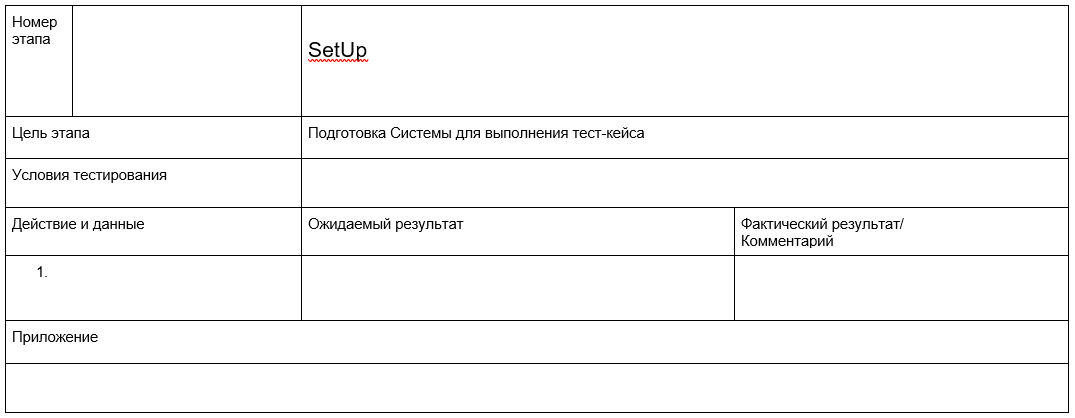


Рисунок 23 - Шаблон для теста[[23]](#footnote-23)

В разделе «Откат изменений» указывается алгоритм действия для отката изменений. В тестах уместно сразу же откатывать изменения, например:

1. Добавить комментарий на график;
2. Проведение проверки что комментарий отображается корректно;
3. Удалить комментарий.

Это была проверка функционала добавление функционала на график и его удаление. Но возможна ситуация, когда откат изменений может повлиять на группу тестов.

Группа тестов представляет собой некоторое количество взаимосвязанных между собой тестов. Их характеризует определенная последовательность между собой и необходимость наличия результатов предыдущего теста.

## 3.2 Формула заполнения тест-плана

### 3.2.1 Требования к заполнению поле «Действие»

Выработана общая формула заполнение поля «Действие» для тест-плана:

* Где это находится;
* Какое действие совершаем;
* На что это действие направленно;
* Описание.

«Где это находится» - указывается если нахождение элемента управления может быть не очевидно для пользователя, который видит интерфейс первый раз:

* В левой боковой форме;
* В ячейке таблицы;
* В поле;
* На вкладке и т.д.

«Какое действие совершаем» - стандартизуется описанием взаимодействия с интерфейсом:

* Нажать, если необходимо нажать на элемент больше одного раза, то указывается количество: «Нажать Х раз»;
* Переместить;
* Ввести;
* Очистить;
* Выделить и т.д.

«На что это действие направленно» - описываются основные элементы управления и взаимодействие для создания единого языка описания интерфейсов программного обеспечения:

* Кнопка;
* Поле;
* Чекбокс;
* Фото и т.д.

«Описание» - содержит описание объекта для его уточнения.

### 3.2.2 Дополнения к заполнению поля «Действие»

При работе с полем даты внутри теста, которые не занимается проверкой функционала часов или календаря указывается полная дата без указания промежуточных шагов перелистывания циферблата часов или смена месяца и года.

Выработанные заметки для заполнения:

* Описание элемента чаще всего находится на кнопке;
* Описание полей находится в непосредственной близости с полем; сверху или снизу.
* Описание формы происходит исходя из модуля в которой расположена форма;
* Единообразие заполнения тест-планов – при наработке одного и того действия применяется стандартная процедура описания.

При работе со списками внутри теста, который не занимается проверкой работы раскрывающего списка описывать взаимодействие без промежуточных шагов.

## 3.3 Использование ментальных карт

Ментальная карта – это метод графической структуризации концепции в виде диаграммы. Диаграмма представляет собой древовидную структуру. Ментальная карта описывается в виде карты зависимостей, где каждый блок — это отдельный тест, которые проверяется определенные функционал. Эта диаграмма позволяет исключить дублирование в тестах.

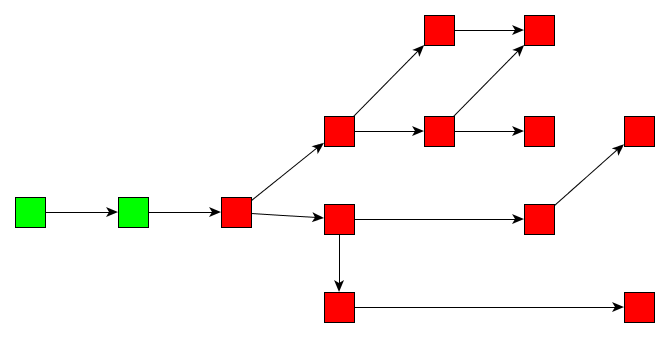


Рисунок 24 - Ментальная карта без атомарности тестирования и точек входа[[24]](#footnote-24)

Основной рекомендации по составлению тест-плана является описание ментальной карты. Главной идеей для составления тестов является покрытий и проверка функционала, для этого необходимо разбираться какой функционал тестируется. Визуализация концепции, представленной на рисунке 26 дает информацию распределении одного и того же теста в разных тест-кейсах. А также ментальная карта дает представления о различных «входах» в различные тестовые среды, т.е. исключается проблема прохождения однотипных тестов по корню дерева, представлено на рисунке 24. При использовании конфигурации тестирования по средством подкладывания информации о необходимом состоянии сессии, проваленные тесты не будет давать неопределенность на остальные тесты, представлено на рисунке 25.

При определенной атомарности теста можно выбирать тест из всего возможно тестового набора. Из общего хранилища всей тестовой информации или окружения наполняется тест в зависимости от контекста его использования и наполненности информацией.

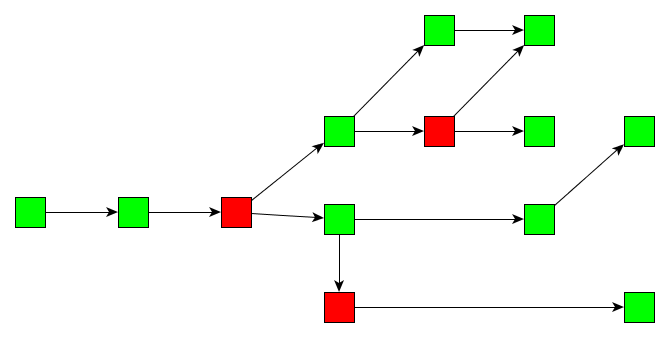


Рисунок 25 - Ментальная карта с использованием точек входа[[25]](#footnote-25)

В описании ментальных карт заинтересованы:

* Инженеры тестирования;
* Аналитики;
* Разработчики программного обеспечения.

Это обуславливается визуализацией системы в целом, что может давать положительный эффект для понимая в какую ветвь системы разрабатывается или в каком месте исправляется функционал.

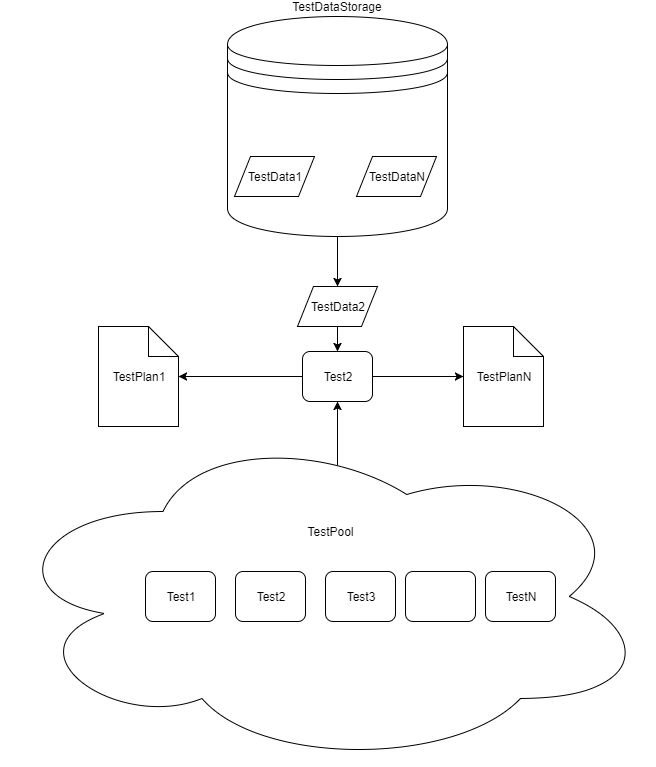


Рисунок 26 - Концепция использования ментальных карт[[26]](#footnote-26)

## 3.4 Практическое применение инструмента «Selenium»

### 3.4.1 Инструкция для работы с «Selenium»

Для массового внедрения автоматизации была разработана инструкция по первоначальной установке Selenium. Она состоит из алгоритма действий для первого запуска среды разработки. В инструкции учтены подготовительные этапы, а именно установка окружения разработчика:

* Java SE Runtime Enviroment 8;
* Java SE Development Kit 8;
* IntelliJ IDEA.

Далее устанавливается сборщик проекта и настраиваются переменные среды, а также скачивается и устанавливается сам «Selenium». В самом конце происходит проверка корректности установки. Указывается ряд причин возможных неисправностей во время установки и пути их решения.

### 3.4.2 Описание работы с «Selenium»

Selenium это программная библиотека для управления браузерами. Представляет собой семейство драйверов для различных браузеров с комплектом клиентских библиотек для работы своего функционала на различных языках программирования.

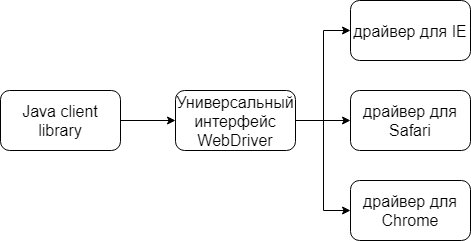


Рисунок 27 - Общее представление Selenium[[27]](#footnote-27)

Текущая версия Selenium 3.12.0 предоставляет возможность напрямую использовать команды браузера, используя стандартный для определенного браузера интерфейс взаимодействия. А также есть возможность внедрения JavaScript код в браузере при запуске для дальнейшего его использования. Тем самым достигается что взаимодействие с браузером происходит наиболее приближено к действиям пользователя.

### 3.4.3 Основные термины для «Selenium»

К основным терминам относятся:

* WebDriver – сущность, которая отвечает за управление браузером. Основное прохождение скриптов происходит при помощи экземпляра этой сущности;
* WebElement – представляет собой абстракцию над элементом интерфейса, включает в себя методы для имитации взаимодействия пользователя с интерфейсов и предоставление их текущего состояния;
* By – абстракция над локатором элемента интерфейса, включающий в себя данные о селекторе и месте нахождения определенного элемента на странице.

### 3.4.4 Применение основного функционала «Selenium»

При автоматизации с помощью «Selenium» инженер автотестирования основываясь на тест-плане описывает алгоритм действий локаторами. На основании стандартных процедур применимых к определенным элементам создает имитацию взаимодействия пользователя.

Таблица 12. Соотношение действий пользователя из теста с локаторами «xpath»

|  |  |  |
| --- | --- | --- |
| № | Действие | Локатор |
| 1 | Нажать на аватар пользователя | //div[1]/div[2]/button[@id=’image’] |
| 2 | Нажать на кнопку редактирования | //button[@id=’correct’] |
| 3 | Ввести в поле «Статус» сообщение о текущем статус | //input[@text=’status’] |
| 4 | Нажать кнопку «ОК» | //button[@id=’confirm’] |
| 5 | Нажать кнопку «Сохранить» | //button[@id=’save’] |

Затем подбирается оптимальные подход к ожиданию элемента, т.к. имитирующий скрипт воспроизводит действия пользователя достаточно быстро. Существуют три основных подхода к ожиданию элемента:

* Явное ожидание, где указывается время простоя в миллисекундах. Основными минусами является, то что повышается минимальное время прохождения теста. Скрипт теста ожидает наступления определенного события перед тем как продолжить выполнять остальные команды;
* Неявное ожидание – Implicit Waits. Этот тип настраивает экземпляр WebDriver на многократные попытки определить веб-элемент на странице в течении выставленного времени, по истечении появится ошибка «ElementNotFoundException». Чаще всего используется при создании экземпляра WebDriver для того чтобы настройки действовали на протяжении его жизни, а также при ожидании отработки скриптов веб-страницы.

На рисунке 28 представлен код для выполнения теста, которые реализует проверку открытия окна браузера и перемещения по странице.

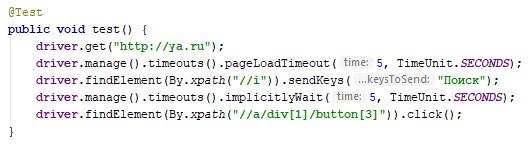


Рисунок 28 - Тест открытия страницы[[28]](#footnote-28)

## 3.5 Использование «Selenium Grid»

После покрытия части системы автотестами появился вопрос о масштабируемости инструментов тестирования. Selenium Grid – это кластеров Selenium-серверов, представленный на рисунке 29.

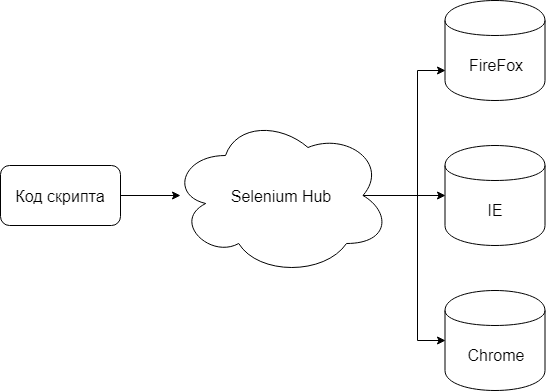


Рисунок 29 - Представление Selenium Grid[[29]](#footnote-29)

Кластер серверов решает ряд задач:

* Распределение скриптов по нескольким тестовым нодам, при помощи параллельного выполнения достигается масштабируемость;
* Конфигурация различных сред в одном инструменте, облегчая проведения тесов на разнообразных комбинациях операционных систем и браузеров;
* Сокращает время на обслуживание сервера, т.к. предоставлена возможность создания пользовательских настроек для конфигурации виртуальной инфраструктуры узла кластера.

Для настройки сети необходимо развернуть Hub, а именно центральный узел управления, который принимает все запросы. Преобразование происходит за счет перераспределения тестового скрипта на заранее смоделированною Node, на которой имитируются действия пользователя на понятном для браузера языке. Из особенностей Hub можно выделить вынос настроек для удобной конфигурации, представленных на рисунке 30.

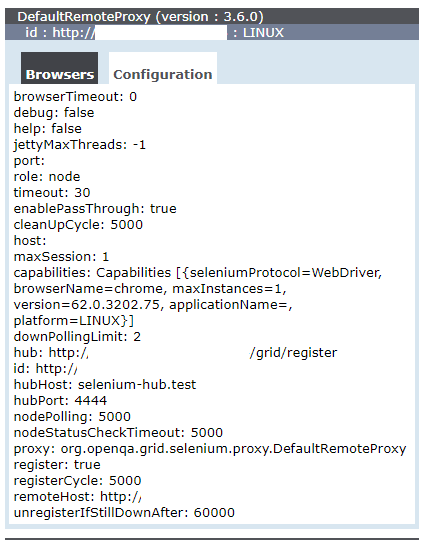


Рисунок 30 - Представление Chrome ноды в распределенной тестовой сети Selenium Grid[[30]](#footnote-30)

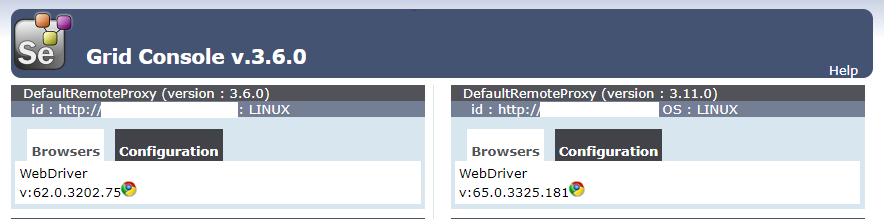


Рисунок 31 - Общее представление Selenium Grid[[31]](#footnote-31)

При нагрузке на Selenium Grid занятые Node будет подсвечиваться серым, по умолчанию максимальное количество параллельных тестов равно пяти. Для изменения настроек можно передавать с дополнительным ключом необязательные настройки. И как принято в настоящее время Hub обладает штатными средствами для диагностики при использовании аномальных наборах параметров.

Таблица 13. Описание возникающих ошибок при работе с Selenium Grid

|  |  |
| --- | --- |
| Причина | Уточнение |
| TIMEOUT | Сессия истекла, т.к. клиент не обращался к ней в течение определенного промежутка времени |
| ORPHAN | Клиент, ожидающий в очереди, отказался от новой сессии |
| CLIENT\_STOPPED  \_SESSION | Клиент остановил сессию, но запрос остановки утерян. |
| CLIENT\_GONE | Процесс клиента не может быть найден |
| FORWARDING\_TO\_  NODE\_FAILED | Нет связи с узлом |
| CREATIONFAILED | Узел не смог запустить браузер |
| PROXY\_REREGISTRATION | Отменена сессия из-за перерегистрации кластера |

## 3.6 Использование docker контейнера

Поскольку кластер состоит из Hub и Node, на практике было выявлено ряд проблем:

* Hub – является единственной точкой доступа к браузерам. Браузеры становятся недоступны, если происходит ошибка, которая приводит к нестандартному завершению процесса, то Hub перестает отвечать;
* Выяснено что один Hub может работать не больше, чем с 20 Node. При максимальной нагрузке запросы обрабатываются слишком медленно и может оборваться связь с Hub.

Основной идей было переход на контейнеры. Контейнеры содержат тоже самое что и внутри Node, а Hub можно заменить легковесный Linux дистрибутивом. Как итог, можно получить небольшие легковесные контейнеры.

В настоящее время самым популярной контейнерной платформой является Docker. Создатели Selenium дают готовый набор контейнеров для запуска Selenium в режиме Grid. По сколько для того чтобы запускать кластер контейнеров необходимо запускать и останавливать образы в ручном режиме было принято решения использовать автоматические сборочные решения Jenkins, рисунок 32.

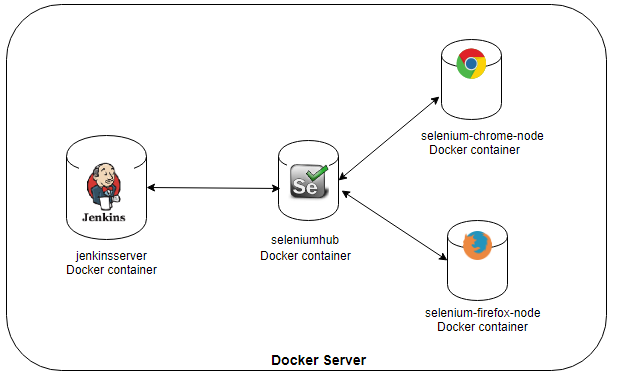


Рисунок 32 - Docker Server[[32]](#footnote-32)

## 3.7 Использование обертки «HtmlElements»

При решении задач тестирования была выявлена проблема, связанная с тем, что при описании одних же элементов WebDriver возникало дублирование кода. По сколько скрипты не предполагают зависимости между собой, только если не находятся в тестовых группах, происходило излишнее описание повторяющихся объектов.

Для решения этой проблемы было принято решения использовать обертку HtmlElements для библиотеки Selenium [7]. Главная идея обертки — это использования подхода Page object [8]. Данный шаблон проектирования позволяет инкапсулировать работу с отдельными элементами веб-страницы, вследствие уменьшается количество кода и его поддержки. Основными преимуществами является:

* Разделение описаний веб-модулей и кода скриптов;
* Все действия по работе с конкретной веб-страницей располагаются в одном месте.

Концепция библиотеки-обертки состоит из:

* Иерархии объектов страницы в виде гибкой композиции java-интерфейсов, т.е. все описание внутри интерфейса;
* Введен дополнительный контроль над поведением веб-элементов с помощью расширенных методов.

## 3.8 Конфигурация запуска тестов с помощью библиотеки TestNG

TestNG – это фреймворк тестирования, дающих возможность писать более гибкие тест.

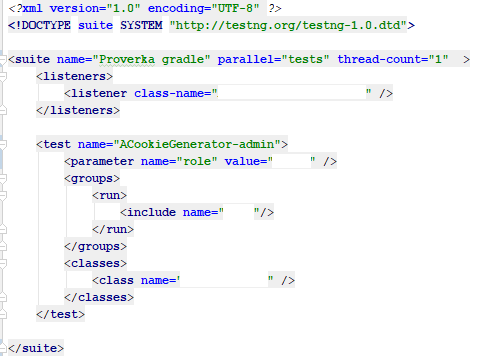


Рисунок 33 - HTML-конфигурация тест-плана с использованием библиотеки TestNG[[33]](#footnote-33)

При взаимодействии с Selenium появляются следующие преимущества:

* Возможность создания HTML-отчетов о результатах тестирования;
* Аннотации очерёдности запуска части кода;
* Группировка связанных между собой тестов;
* Возможность запускать тестов параллельно;
* Параметризация данных.

С помощью этого фреймворка стало возможным конфигурировать тест-планы, выраженные в скрипте в одном месте, а не запускать тесты по отдельности. Передача параметров организует работу различных юзеров на одних и тех тестовых данных.

## 3.9 Использование библиотеки AllureReport

AllureReport это библиотека предоставляющая отчет по тестированию, в котором содержится сжатое представление о том, что было протестировано и позволяет всем участникам разработки извлечь максимум информации.

Отчеты дают наглядное представление о жизненном цикле дефекта. По результатам теста определяется его статус, он имеет несколько вариантов:

* Failed тесты которые не прошли процесс сравнивания фактического и ожидаемого результата;
* Broken тесты которые завершили свою работу не корректно, т.е. не смогли нужный элемент, истекло время ожидания элемента, аварийное завершение работы скриптов на веб-странице;
* Passed тесты которые успешно прошли все этапы сравнений ожидаемого и действительно результата;
* Skipped тесты которые были пропущены, чаще всего происходит из-за очереди на выполнения скрипта или провала группы тестов;
* Unknown тесты по которым по тем или иным причинам информации нет, происходит из-за разрыва связи между сервером Hub и Node.



Рисунок 34 - Агрегированные сведения по результатам тестирования с помощью библиотеки AllureReport[[34]](#footnote-34)

Обычный отчет состоит из вкладки «Обзор», панели навигации, нескольких вкладок для различных видов представления тестовых данных и страниц с тестами. Каждый отчет представлен в виде древовидной структуры, вкладки позволяют переключаться между преставлениями исходной структуры данных, а также присутствует фильтрация и сортировка.

На основной странице располагается:

* Статистика – общая статистика отчета;
* Количество запусков, если было несколько;
* Поведение – информация о результатах, собранная в соответчики с историческими данными и особенностями запуска;
* Исполнители тестов – перечисляются инструменты которые использовались для запуска тестов;
* Тенденция – если тести выполняются несколько раз, то на основании исторических данный будет представлен и рассчитан тренд;
* Тестовой окружение – информация о тестовой среде.

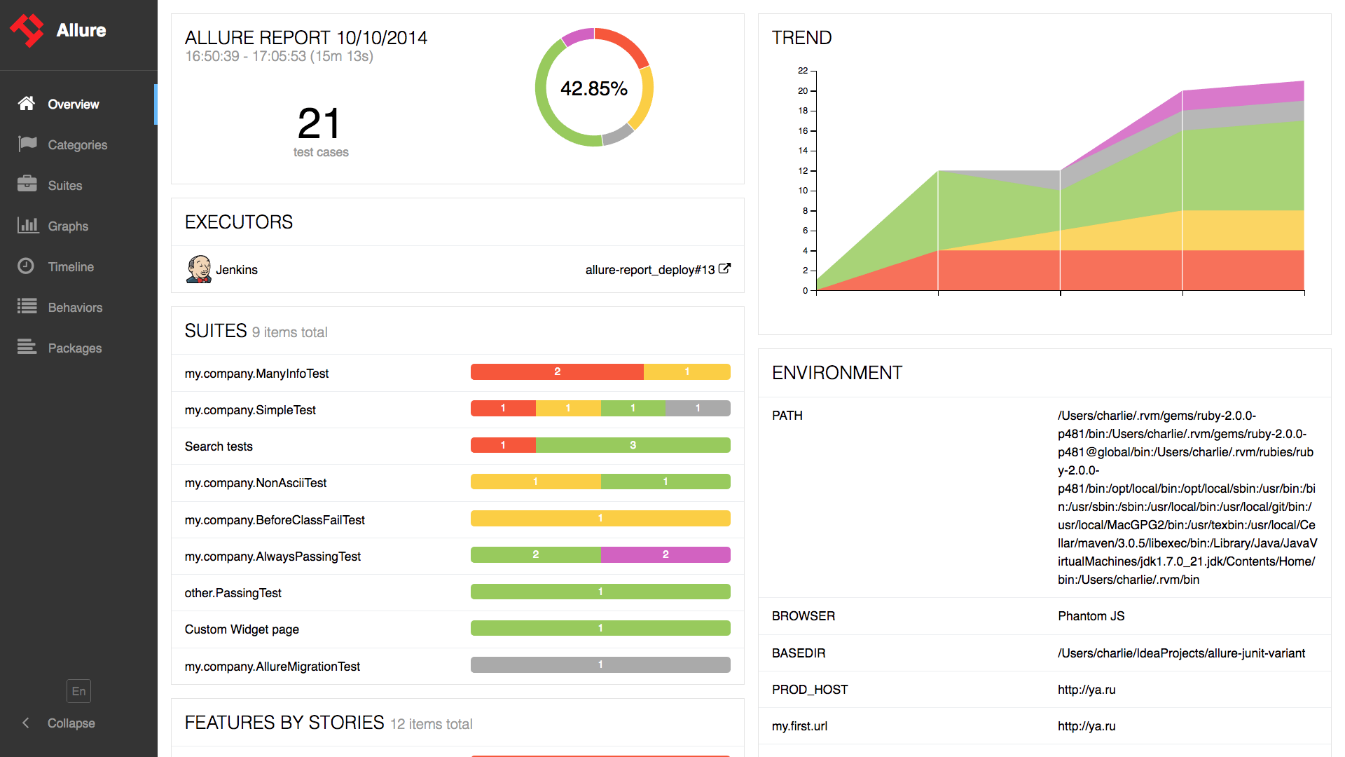


Рисунок 35 - Общий вид AllureReport[[35]](#footnote-35)

## 3.10 Процесс развертывания

Заключительной частью проектирования это встраивание в процесс развертывания. В контексте данной работы применяется Jenkins как оркестрант для распределения контейнеров тестовых Node, процесс развертывания стал выглядеть следующим образом, рисунок 36.

После выполнения скриптов результаты тестирования отправляются по электронной почте всем заинтересованным лицам:

* Разработчик удостоверился что его изменения ничего сломали;
* Аналитик понимает текущее состояние системы;
* Инженер администрирования – следить за процессом развертывания и анализом интеграции;
* Инженер тестирования – контролирует результаты тестирования.

За эту настройку отвечает плагин Jenkins, сообщение содержит время сборки, результат сборки и путь до файлов AllureReport.

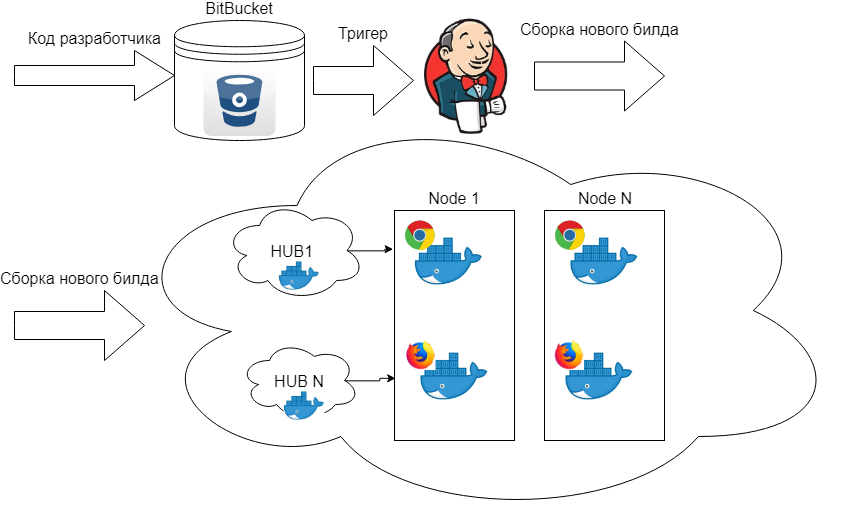


Рисунок 36 - Процесс развертывания[[36]](#footnote-36)

## 3.10 Валидация результатов проектирования

Во время проектирования были применены практические и научные знания в решении задач для создания систематизированного процесса. Полночный процесс представляет собой группу взаимосвязанных элементов для контроля качества.

Разработан подход к ручному тестированию, где учтены основные особенности для удобств заполнения тестировщиками, а также добавлена читаемость для остальных участников процесса. Для хранения тест-планов используется хранилище Confluence, у всех заинтересованных в запланированных сценариях появилась возможность их обсуждать, что благоприятно влияет на распределение мнений внутри команды. Ментальные карты позволили взглянуть на систему максимально детерминировано, разделив программное решения на множество ветвей. Теперь у каждого участника процесса появилось визуальное представление о взаимодействие модулей между собой.

В разделе автоматизации тестирования появился ряд аспектов:

* Фиксация инструмента тестирования:
* Использования новых технологий для улучшения взаимодействия тестируемого приложения;
* Автогенерируемый отчет;
* Интеграция в процесс непрерывной доставки.

Для описания экономической эффективности спроектированной системы будет приведен расчет трудозатрат различных участников процесса разработки.

Без проектируемой системы процесс разработки выглядел следующим образом, от лица бизнес-аналитика:

* Детализация требований и постановка задач;
* Передача задач разработчикам;
* После накопления определенного набора задач необходимо продемонстрировать заказчику промежуточный результат. Для этого бизнес-аналитик описывал инструкции по эксплуатации на основе своих же постановок задач;
* После демонстрации возвращался и детализировал замечания полученный от заказчика для дальнейшего исправления.

В данном случаем система верификации позволяет до того, как бизнес-аналитик начнет описывать инструкции спроектировать ментальные карты. Это экономит время для технического описания системы в целом и позволяет не пропустить связи между ветвями. А также затрагивается вопрос того, что более дорогой для компании сотрудник перед презентацией проверяет работоспособность функционала. В случае же предварительно настроенного автотестирования, стейкхолдер может предоставить ряд пожеланий для проверки и получить отчет о состоянии нового функционала.

От лица разработчика программного обеспечения:

* После постановки задачи от бизнес-аналитика разработчик приступает к процессу кодирования;
* Отправляет промежуточное решение в хранилище
* При неисправности тратить время на исследовательские задачи.

Аналогично предыдущему случаю ментальная карта может помочь разработчику разобраться с нетривиальной связи комплексной системы, что может ощутимо влиять на качество разработки. После отправки промежуточного решения в облако нет представления о статус сборки проекта, в случае же автотестирования по результатам успешной или не успешной сборки будет получен автогенерируемый отчет. Время на воспроизведение ошибки возможно сократить при регламентированном подходе к описанию проблем.

На рисунке 37 представлена круговая диаграмма общих затрат на содержание команды разработки в месяц, состоящей из двух бизнес-аналитиков, шести разработчиков, двух ручных тестировщиков, одного инженера по автоматизации тестирования и инженера администрирования.

Поскольку основные затраты на содержание уходят на разработчиков, поэтому целесообразно как можно более эффективно использовать этот ресурс, а именно не вынуждать заниматься исследовательскими задачами в локализации ошибок. А также предоставлять им информацию о состоянии системы в целом с помощью ментальных карт, т.к. часто разработчик, находясь в своем модуле может быть не осведомлен о изменения ветвей, это может предотвратить ошибки при взаимодействии компонентов.

Таблица 14 - Среднерыночное описание зарплат команда разработки

|  |  |
| --- | --- |
| Позиция | Средняя заработная плата, руб. |
| Аналитик | 150000 |
| Разработчик | 100000 |
| Ручной тестировщик | 45000 |
| Инженер автотестирования | 70000 |
| Инженер администрирования | 90000 |

Рисунок 37 - Месячные затраты на содержание команды разработки[[37]](#footnote-37)

Аналогично с аналитиками, их необходимо также разгрузить с помощью новых возможностей, что приведет к положительному экономическому эффекту. Происходит более эффективное перераспределение ресурсов за счет передачи задач более низкооплачиваемому персоналу.

Но в случае с инженером автотестирования можно заметить более ощутимый вклад в стоимость разработки в месяц. При этом заметная польза в виде отчетов и настройки регрессионного тестирования предполагается что будет не сразу. На рисунке 38 представлена основная зависимость между стоимостью поддержания автотестирования и ручного тестирования.

Рисунок 38 - Сравнение затрат на содержание ручного и автотестирования[[38]](#footnote-38)

При достижении определенного уровня насыщения процесс автотестирования становится менее затратен с течением времени, но при старте является более дорогим решением. Автоматизация рассматривается как долгосрочная перспектива развития с повышенным начальным капиталом для выстраивания процессов, рисунок 39.

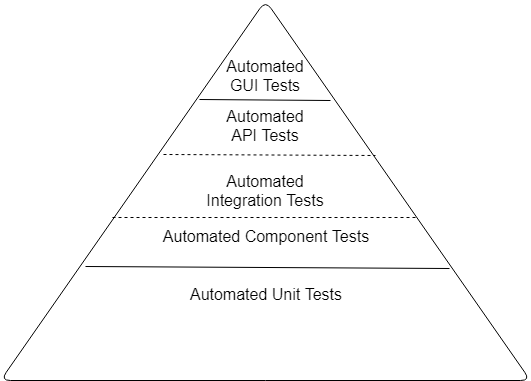


Рисунок 39 - Перспективы изменения пирамиды тестирования[[39]](#footnote-39)

# ЗАКЛЮЧЕНИЕ

Наличие потребности всех затронутых участников в системе верификации программного обеспечения доказывает актуальность рассматриваемой темы и практическую применимость спроектированной системы.

Проведен этап верхнеуровневого анализа бизнес-процесса разработки программного обеспечения, что является основанием для дальнейшего анализа и реинжиниринга бизнес-процессов компании.

При реализации проекта были выявлена потребности стейкхолдеров, выставлены к ним требования и после этого предложены, а также проанализированы варианты решений:

* регламентированная работа с тест-кейсами,
* стандартизация ключевых слов при заполнении теста и отчета об ошибке,
* описана методология использования ментальных карт,
* собраны требования для инструмента автотестирования,
* выбран и апробирован инструмент автотестирования.

А также дополнительные технологии для улучшения процесса тестирования в виде:

* Использованием шаблона PageObject;
* Конфигурация запуска тестов;
* Визуализация отчетности.

В ходе работы были применены следующие методики системного анализа:

* Определение системного анализа
* Исследование модели бизнес-процесса
* Инженерия требований
* Генерация решений
* Анализ решений
* Обзор аналог и т.д.

Были применены компетенции в области:

* По проектированию фреймворка тестирования;
* Моделирование в среде ArchiMate
* Разработка коммерческого кода внутренних решений поддержки тестирования с использование языка программирования Java.

Выполнено системно-инженерное описание проекта, основанное по результатам альф:

* Стейкхолдеры – представлены потребности и требование заинтересованных сторон, декомпозиция потребностей с уровня бизнес-целей до уровня технических требования и реализация, выявлены основные проблемы;
* Возможности – представлена концепция системы верифкации, проведена ее апробация в рамках процесса разработки «ZOZO WFM» и валидация решения на уровне сбора обратной связи от стейкхолдеров
* Работы – работа по проектированию и разработке согласно требованиям заинтересованных сторон;
* Технологии – основные технологии описаны в рамках работы. Главные идеи проектирования раскрыты, но частичное описание процессов необходимо в целях сохранения коммерческой информации.

Работа в рамках завяленной темы выполнена полностью, несет практическую ценность как для заказчика системы, так и для других стейкхолдеров.

# БИБЛИОГРАФИЧЕСКЙ СПИСОК

1. Системная инженерия. Принципы и практика / Косяков Алексаднр, Уильям Н. Свит, Семюэль Дж. Сеймур, Стивен М. Бимер. – Москва : ДМК Пресс, 2014. – 624 с.
2. V-Model // Википедия. [2018—2018]. Дата обновления: 18.05.2018. URL: <https://ru.wikipedia.org/?oldid=92728569> (дата обращения: 18.05.2018).
3. ISO 9000:2015 Quality management systems — Fundamentals and vocabulary – Part 3: Terms and definitions.
4. Качество программного обеспечения // Википедия. [2018—2018]. Дата обновления: 01.03.2018. URL: https://ru.wikipedia.org/?oldid=91243141 (дата обращения: 01.03.2018).
5. Stakeholder Needs and Requirements. (2015, June 29).  *SEBoK,*. Retrieved 11:29, June 12, 2018  from <http://www.sebokwiki.org/w/index.php?title=Stakeholder_Needs_and_Requirements&oldid=51430>.
6. Тестирование в Яндексе. Как сделать отказоустойчивый грид из тысячи браузеров [Электронный ресурс]. – Режим доступа: <https://habr.com/company/yandex/blog/268309/>, свободный (дата обращения 10.05.2018)
7. [Электронный ресурс]. – Режим доступа:  <https://github.com/eroshenkoam/htmlelements> , свободный (дата обращения 10.04.2018)
8. [Электронный ресурс]. – Режим доступа: <https://kreisfahrer.gitbooks.io/seleniumwebdriver/content/page_object_pattern_arhitektura_testovogo_proekta/ispolzovanie_patterna_page_object.html> , свободный (дата обращения 20.04.2018)

# ПРИЛОЖЕНИЕ А

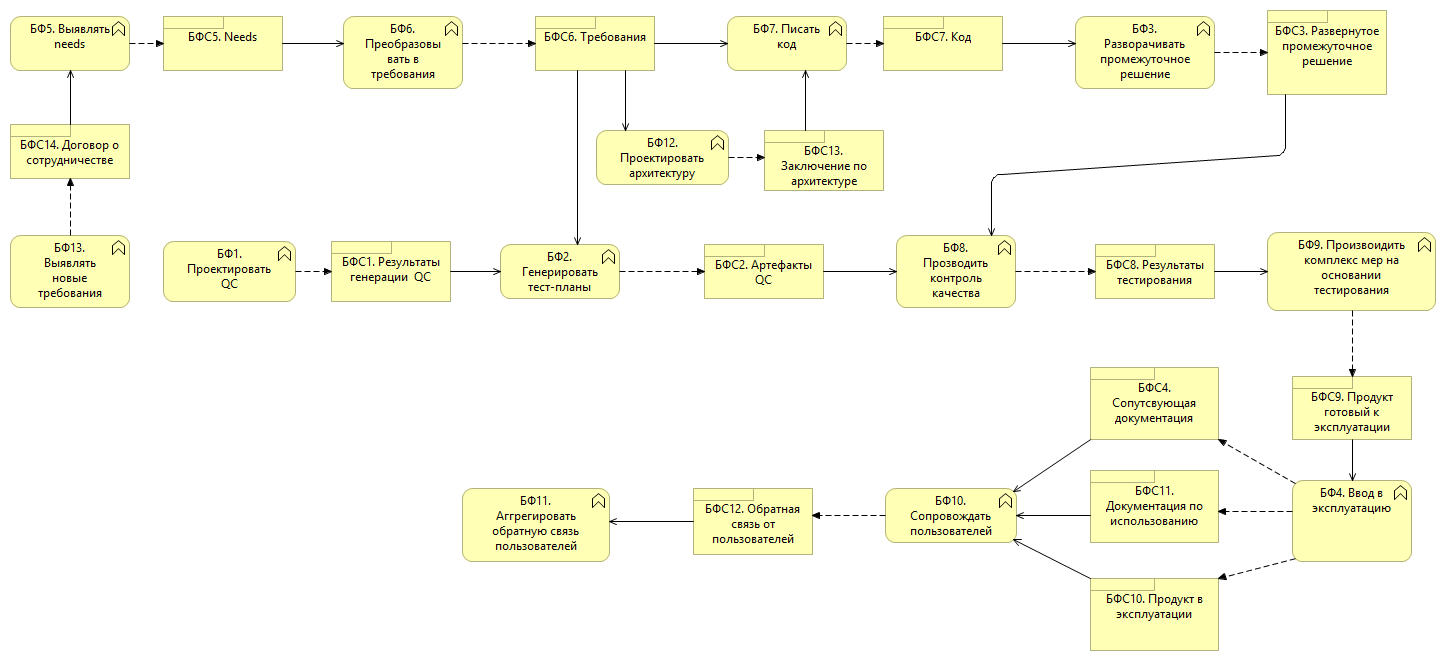


Рисунок А1 – Функциональная модель бизнес-контекста

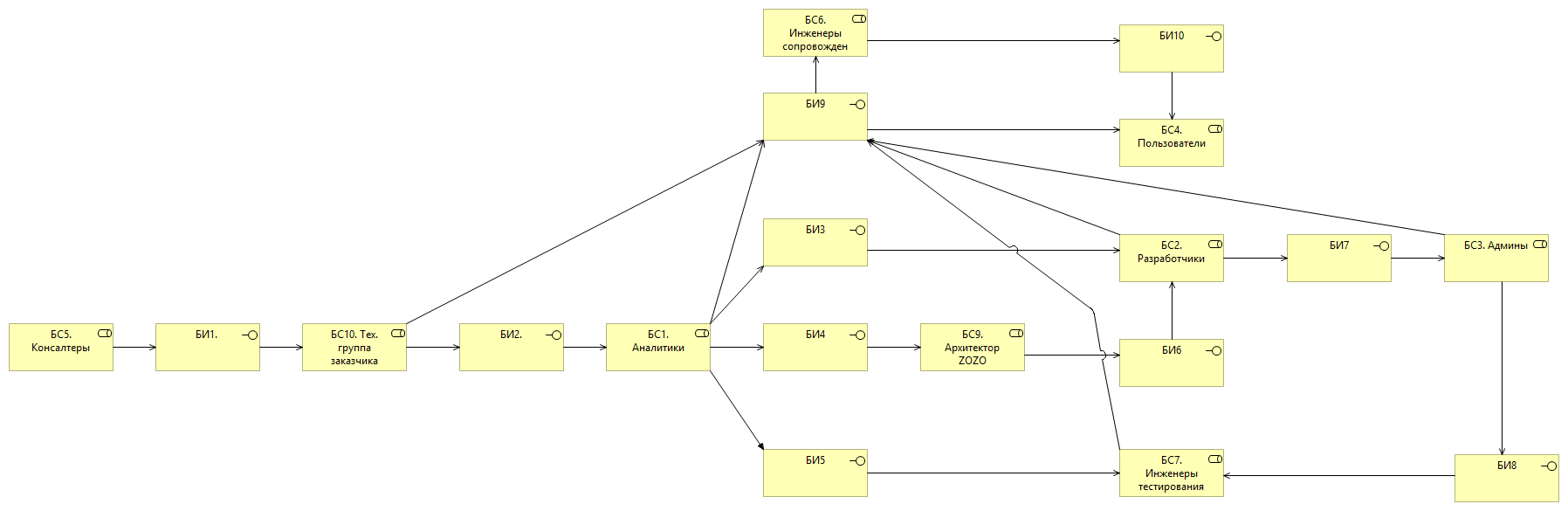


Рисунок А2 – Структурная модель бизнес-контекста

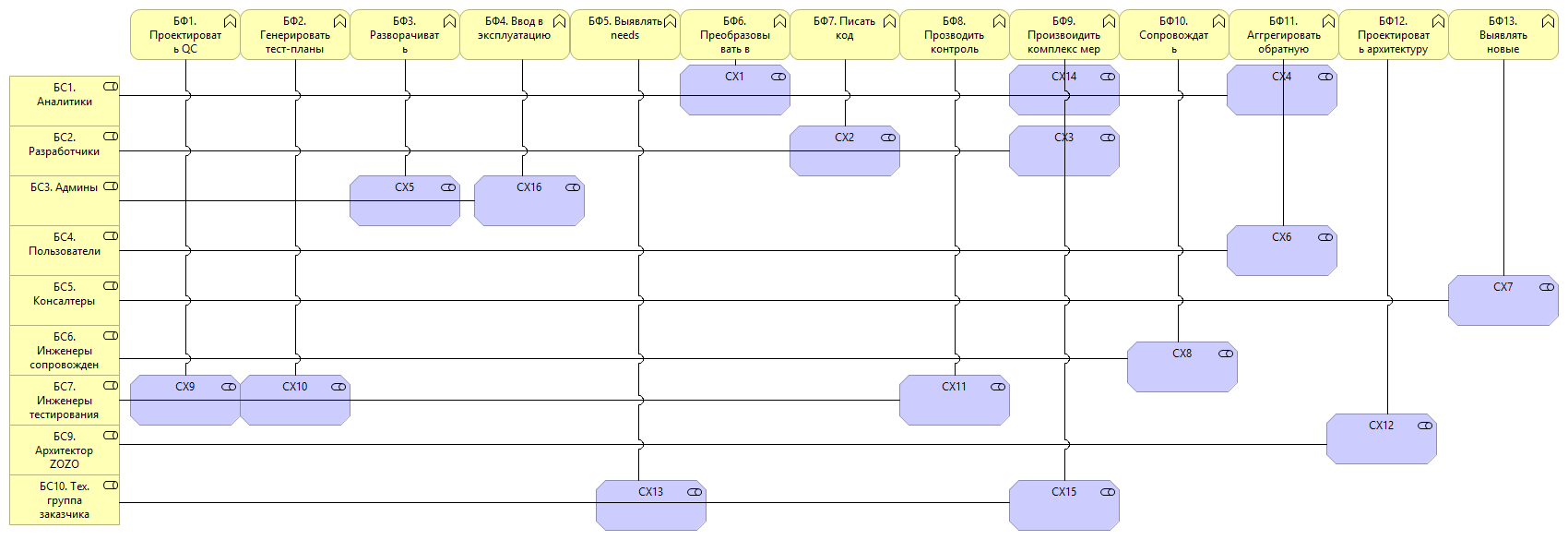


Рисунок А3 – Матрица определения стейкхолдеров

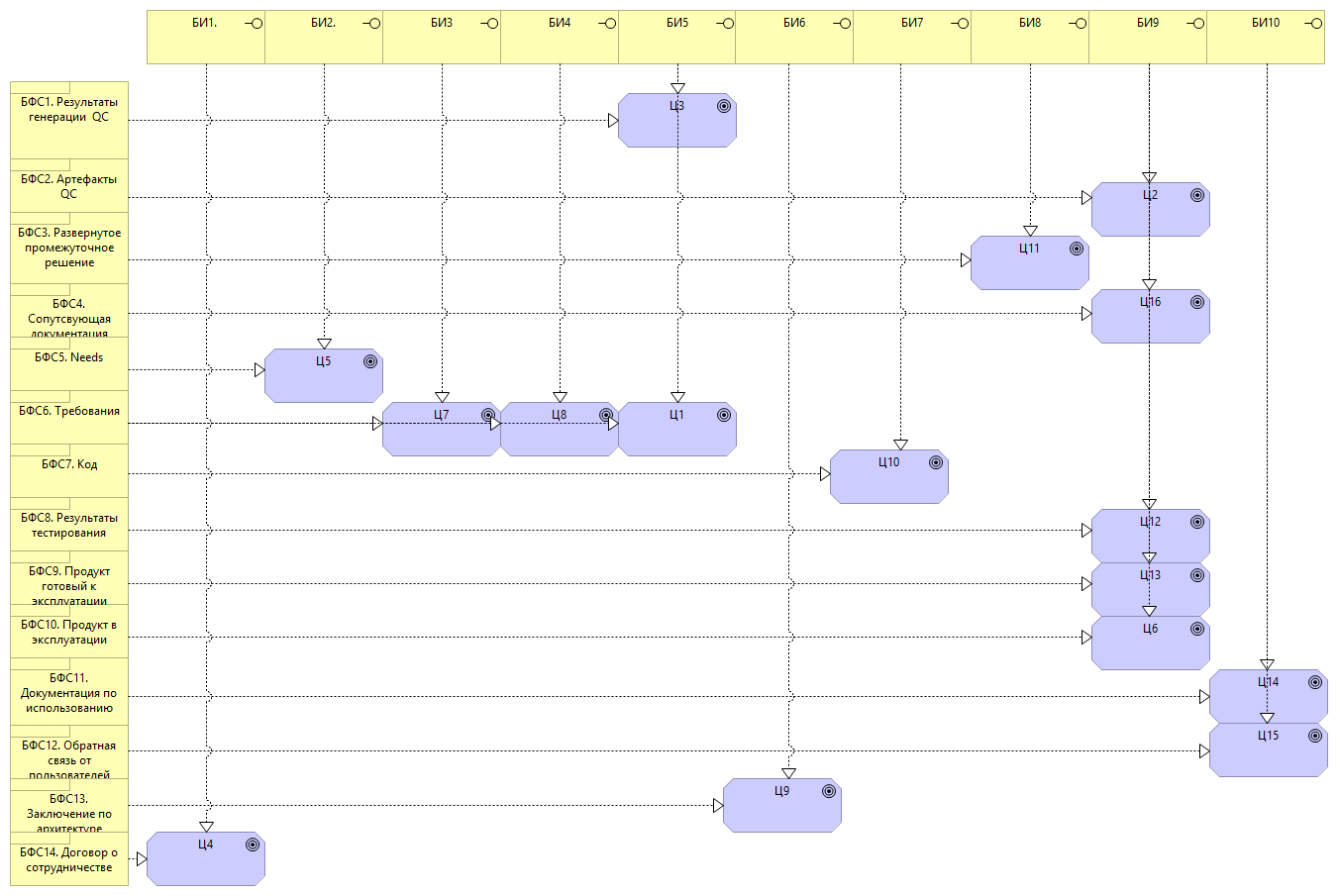


Рисунок А4 – Матрица определения целей

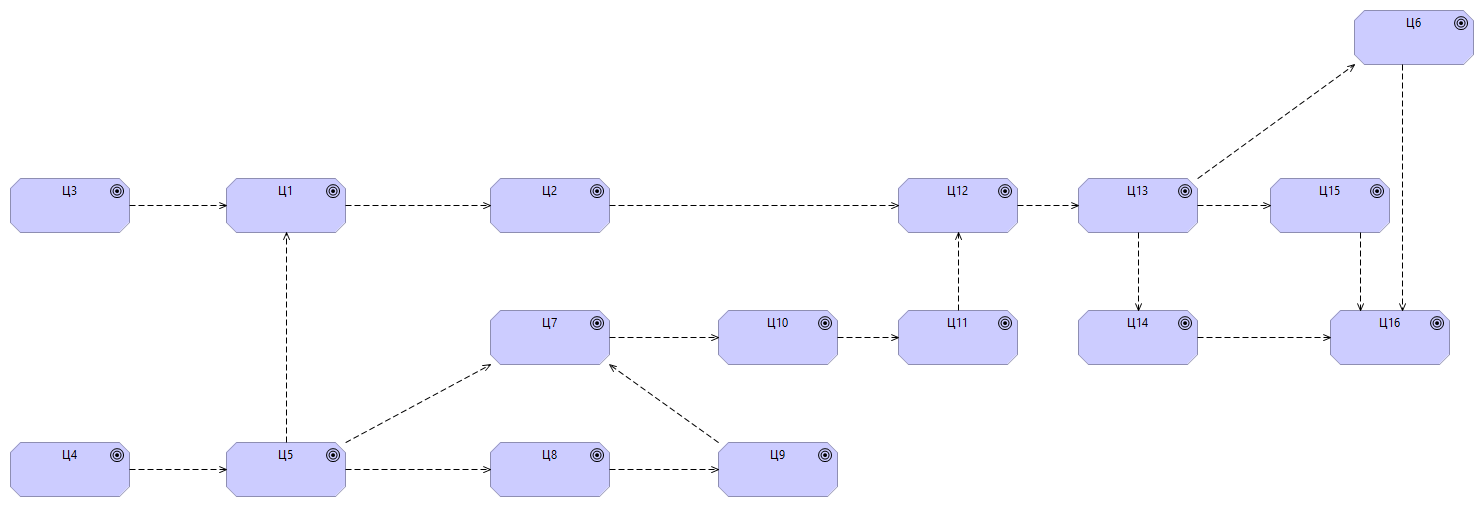


Рисунок А5 – Схема взаимосвязи между бизнес-целями

1. Выполнено автором с помощью yEd 3.18.0.2 [↑](#footnote-ref-1)
2. Выполнено автором с помощью draw.io v8.8.1 [↑](#footnote-ref-2)
3. Выполнено автором с помощью draw.io v8.8.1 [↑](#footnote-ref-3)
4. Выполнено автором с помощью draw.io v8.8.1 [↑](#footnote-ref-4)
5. Выполнено автором с помощью draw.io v8.8.1 [↑](#footnote-ref-5)
6. Выполнено автором с помощью draw.io v8.8.1 [↑](#footnote-ref-6)
7. Выполнено автором с помощью Archi 4.1.1 [↑](#footnote-ref-7)
8. Выполнено автором с помощью Archi 4.1.1 [↑](#footnote-ref-8)
9. Выполнено автором с помощью Archi 4.1.1 [↑](#footnote-ref-9)
10. Выполнено автором с помощью Archi 4.1.1 [↑](#footnote-ref-10)
11. Выполнено автором с помощью draw.io v8.8.1 [↑](#footnote-ref-11)
12. Выполнено автором с помощью Archi 4.1.1 [↑](#footnote-ref-12)
13. Выполнено автором с помощью Archi 4.1.1 [↑](#footnote-ref-13)
14. Выполнено автором с помощью Archi 4.1.1 [↑](#footnote-ref-14)
15. Выполнено автором с помощью Microsoft Excel 16.0.9330.2087 [↑](#footnote-ref-15)
16. Выполнено автором с помощью Microsoft Excel 16.0.9330.2087 [↑](#footnote-ref-16)
17. Выполнено автором с помощью Microsoft Excel 16.0.9330.2087 [↑](#footnote-ref-17)
18. Выполнено автором с помощью Microsoft Excel 16.0.9330.2087 [↑](#footnote-ref-18)
19. Выполнено автором с помощью Microsoft Excel 16.0.9330.2087 [↑](#footnote-ref-19)
20. Выполнено автором с помощью Microsoft Excel 16.0.9330.2087 [↑](#footnote-ref-20)
21. Выполнено автором с помощью Microsoft Excel 16.0.9330.2087 [↑](#footnote-ref-21)
22. Выполнено автором с помощью Microsoft Word 16.0.9330.2087 [↑](#footnote-ref-22)
23. Выполнено автором с помощью Microsoft Word 16.0.9330.2087 [↑](#footnote-ref-23)
24. Выполнено автором с помощью yEd 3.18.0.2 [↑](#footnote-ref-24)
25. Выполнено автором с помощью yEd 3.18.0.2 [↑](#footnote-ref-25)
26. Выполнено автором с помощью draw.io v8.8.1 [↑](#footnote-ref-26)
27. Выполнено автором с помощью draw.io v8.8.1 [↑](#footnote-ref-27)
28. Выполнено автором с помощью IntelleJIDEA 2017.2.7 [↑](#footnote-ref-28)
29. . Выполнено автором с помощью draw.io v8.8.1 [↑](#footnote-ref-29)
30. [Электронный ресурс]. – Режим доступа: https://selenium2.ru/docs/selenium-grid.htmlсвободный (дата обращения 20.04.2018) [↑](#footnote-ref-30)
31. [Электронный ресурс]. – Режим доступа: https://www.seleniumhq.org//, свободный (дата обращения 20.04.2018) [↑](#footnote-ref-31)
32. Выполнено автором с помощью draw.io v8.8.1 [↑](#footnote-ref-32)
33. Выполнено автором с помощью IntelleJIDEA 2017.2.7 [↑](#footnote-ref-33)
34. Выполнено автором самостоятельно [↑](#footnote-ref-34)
35. [Электронный ресурс]. – Режим доступа: <https://docs.qameta.io/allure/> , свободный (дата обращения 15.02.2018) [↑](#footnote-ref-35)
36. Выполнено автором с помощью draw.io v8.8.1 [↑](#footnote-ref-36)
37. Выполнено автором с помощью Microsoft Excel 16.0.9330.2087 [↑](#footnote-ref-37)
38. Выполнено автором с помощью Microsoft Excel 16.0.9330.2087 [↑](#footnote-ref-38)
39. Выполнено автором с помощью draw.io v8.8.1 [↑](#footnote-ref-39)