Федеральное государственное автономное образовательное учреждение

высшего образования

"Российский университет транспорта" (МИИТ)

Кафедра «Цифровые технологии управления транспортными процессами»

**ОТЧЁТ**

**ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

**Тип практики:** ПРОИЗВОДСТВЕННАЯ ПРАКТИКА

(технологическая - 1)

“Анализ и изучение тестирования ПО, его подвидов и варианты применения”

Выполнил ст. г. УВА-411 Романов Д.Н,

Принял руководитель: Н. М. Нечитайло

Москва  
2021

|  |
| --- |
| УТВЕРЖДАЮ  Руководитель ОП ВО по  направлению 09.03.01  \_\_\_\_\_\_\_\_ Э.К. Лецкий  \_\_.07.2021 г. |

**ЗАДАНИЕ НА ПРАКТИКУ**

**производственную практику**

**(технологическая - 1)**

студента группы УВА-411 Романова Даниила Николаевича

Место прохождения производственной практики: кафедра ЦТУТП ИУЦТ РУТ (МИИТ)

Наименование темы: Анализ и изучение тестирования ПО, его подвидов и варианты применения.

Необходимо изучить:

1. Тестирование ПО
2. Ручное тестирование
3. Автоматизированное тестирование
4. Selenide, Selenoid, Cucumber

Необходимо самостоятельно выполнить:

1. Разбор элемента жизненного цикла разработки ПО “Тестирование”
2. Разбор процесса автоматизированного тестирования
3. Анализ решений крупных компаний
4. Анализ лучших решений

Основные требования к проделываемой работе:

Для оформления отчета по практике должен быть использован ГОСТ 7.32-2017  «Отчет о научно-исследовательской работе. Структура и правила оформления».

Источники информации:

1. Книги.

.Материалы к защите:

1. Заполненный, подписанный и утверждённый лист индивидуального задания на производственную практику.
2. Пояснительная записка (отчёт) по производственной практике.
3. Презентация о проделанной работе.

Студент группы УВА-411 Д.Н. Романов

Профессор кафедры ЦТУТП Н.М. Нечитайло

**Оглавление**

[**1** **Разработка формы авторизации** 3](#_Toc92836973)

[**2** **Проверка введенных данных на корректность введенных значений** 8](#_Toc92836974)

[**3** **Реализовать взаимосвязь между БД и формой авторизации** 10](#_Toc92836975)

[**Заключение** 15](#_Toc92836976)

[**Источники информации** 16](#_Toc92836977)

[**Приложение A Зависимости приложения** 17](#_Toc92836978)

[**Приложение B Конфигурационные файлы** 19](#_Toc92836979)

[**Приложение C Контроллеры** 21](#_Toc92836980)

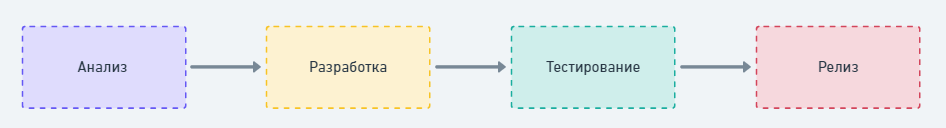
[**Приложениe D html формы** 23](#_Toc92836981)

[**Приложение E сущности** 39](#_Toc92836982)

[**Приложение F DAO** 41](#_Toc92836983)

1. **Вступление**

Современный цикл разработки ПО совмещает в себе несколько простых шагов. Анализ требований, разработка приложения, его тестирование, релиз.



* Анализ. Здесь происходит разговор с заказчиком, выясняются требования, пишется документация. Происходит постановка задачи на разработку, пишется техническое задание (ТЗ).
* Разработка. Большой и непрерывный этап, выполняется реализация ТЗ, дополняются фичи к реализации, корректируется постановка задачи и т.д.
* Тестирование. Непрерывный процесс тестирования приложения, которое находится в разработке, проверяется не только его соответствие требованиям, но и пропускная способность, отказоустойчивость, готовность к непредвиденным ситуациям и к различному поведению пользователей внутри системы.
* Релиз. Непрерывная поставка оттестированного приложения заказчику для уточнения требований, возможных доработок и непосредственной эксплуатации.

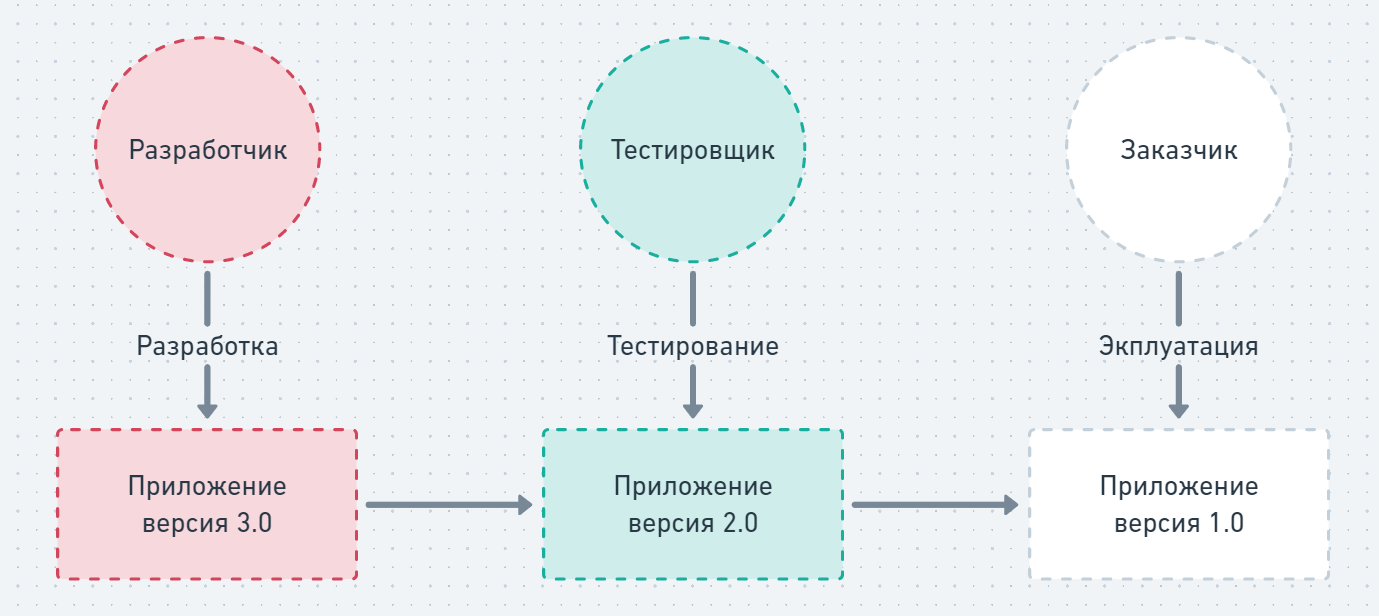
1. **Разбор элемента жизненного цикла разработки ПО “Тестирование”**

Тестирование.



* Модульное тестирование. Эти тесты пишут сами разработчики для проработки маленьких кусочков кода во избежание очевидных ошибок работы кода. Покрывается около 70% кода, все методы и логические ветвления.
* Функциональное тестирование. Здесь в дело вступают тестировщики, производится проверка заявленного функционала.
* Smoke тестирование. Простая проверка, что наше приложение не ломается при попытке его использовать, конечный результат неважен. Успешное smoke тестирование позволяет приступить к другим тестам.
* Интеграционное тестирование (A/B тесты). Позволяет протестировать конкретные тестовые сценарии из точки A в точку B. Задевает сразу несколько функций приложения. Например, авторизация -> создание объектов -> изменение -> выгрузка -> удаление -> выход из приложения.  
  Интеграционный тест считается проваленным, если не выполнился хотя-бы один пункт теста.
* Нагрузочное тестирование. Его цель – сломать приложение под предельными нагрузками, чтобы знать его слабые места и устранить их по итогу.
* Системное тестирование. Такое тестирование покрывает функционал интеграции с другими приложениями. Например, мы добавили в приложение Google.Maps и хотим убедиться, что интеграция с ними работает успешно, наши объекты на карте отображены корректно и функционируют.
* Регрессионное тестирование. Это множество различных тестов, которые были успешными в прошлом релизе. После нового релиза их снова проверяют, чтобы убедиться, что новый функционал не сломал что-то из уже работающего. Позволяет легко выявлять недочёты, которые негативно влияют на старый оттестированный функционал. Количество регрессионных тестов увеличивается с каждым новым релизом.

Рассмотрим схему взаимодействия разработчика, тестировщика и заказчика.



У заказчика всегда самая старая версия приложения. Он пишет пожелания разработчикам, недочёты, новый функционал, излагает идеи.

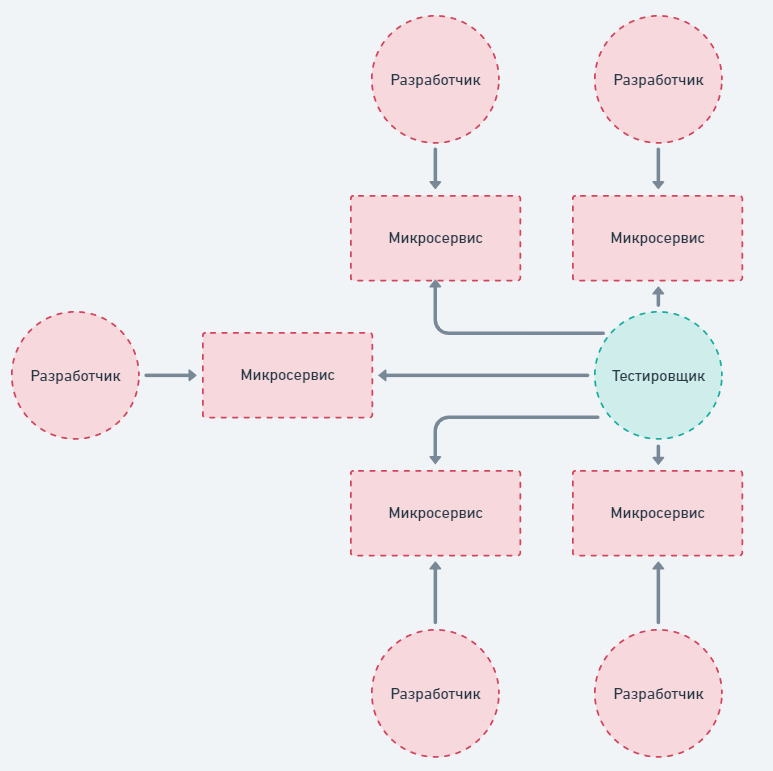
Тестировщик проверяет более свежую версию, проводит функциональное тестирование версии 2.0, регрессионное тестирование функционала старой версии 1.0 и т.д.

Разработчик готовит новую версию и так по кругу.

Проблемы появляются, когда у нас имеется несколько разработчиков и приложение делится на несколько маленьких, которые работают вместе (микросервисная архитектура).

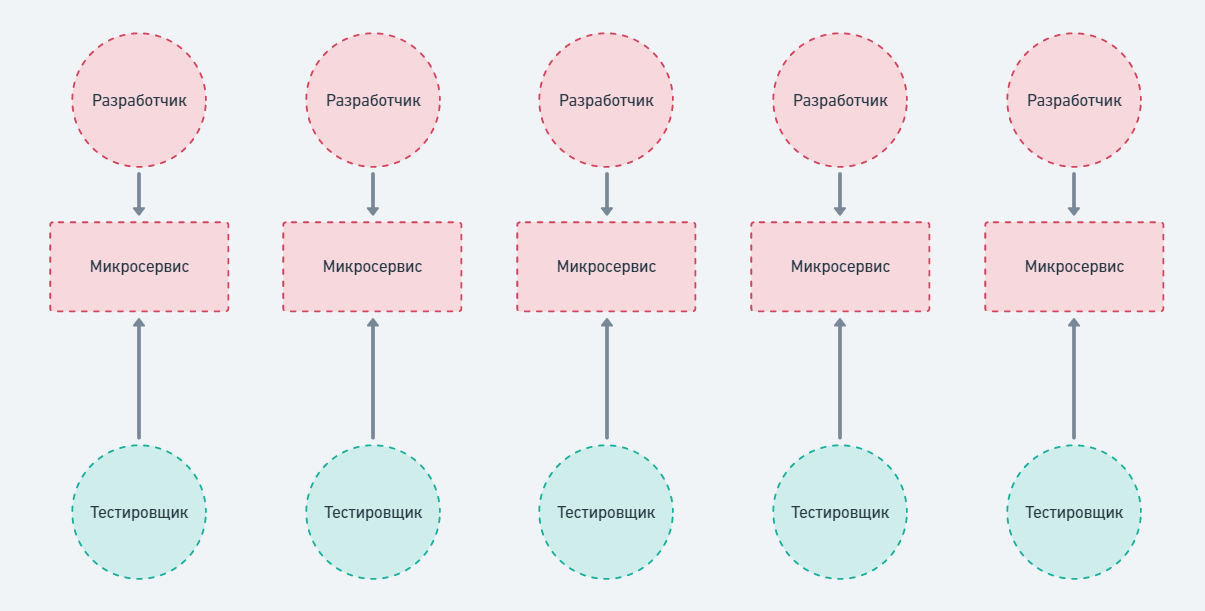
Тогда наша схема преобразуется в следующую.

Например, так.



Получается, что тестировщик перегружен, необходимо либо уменьшить количество разработчиков, что замедлит ход релизов, либо увеличить количество тестировщиков. Увеличим количество тестировщиков.

Получаем следующую схему взаимодействия



Отлично. Теперь релизы будут быстрыми за счёт дополнительных четырёх тестировщиков. Но расходы компании на тестирование возросли в пять раз, что не очень хорошо, ведь если разработчик закончит микросервис, он может приступать к следующему, и тогда нам понадобится ещё один тестировщик на новый микросервис, ведь регрессионное тестирование готового микросервиса никто не отменял, его функционал постоянно проверяется и, возможно, будет дорабатываться.

Такая схема взаимодействия 1 микросервис = 1 тестировщик очень дорогая и нам не подходит. Потому и придумали автоматизированное тестирование.

1. **Разбор процесса автоматизированного тестирования**

**Заключение**

В ходе практики были реализованы формы авторизации и регистрации сервиса для геологической компании. При этом, данные в формах проверяются на корректность, а после выполнения проверок заносятся в БД. А также были изучены современные и мощные инструменты, позволяющие вести разработку web-приложений - Spring Framework и Thymeleaf.

**Источники информации**

1. Герберт Шилдт - Java. Полное руководство; издательство:   
   Диалектика-Вильямс, 10-е издание, 2018г – 1488 с. // [Электронный ресурс] – URL:[Java. Полное руководство - Герберт Шилдт, 10-е издание.pdf (vk.com)](https://vk.com/doc26879026_492639796?hash=2200b3ca37bfd4d34d&dl=2879c460fdaa1d3665). (дата обращения 12.10.2021)
2. [Уоллс Крейг](https://www.labirint.ru/authors/128392/) - Spring в действии; издательство: [ДМК-Пресс](https://www.labirint.ru/pubhouse/1416/), 2015 г -754 с. (дата обращения 14.11.2021)
3. Козмина Юлиана, Харроп Роб, Шефер Крис, Хо Кларенс – Spring 5 для профессионалов; издательство: Диалектика-Вильямс, 5-е издание, 2019г – 1120 с. (дата обращения 13.12.2021)