

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«Дальневосточный федеральный университет»**

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ  
Департамент программной инженерии и искусственного интеллекта**

КУРСОВАЯ РАБОТА

по дисциплине «Технологии коллективной промышленной разработки информационных систем» по образовательной программе подготовки бакалавров по направлению 09.03.04 «Программная инженерия»

Выполнили студенты гр. Б9121-09.03.04  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Карпачев И.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Михайлов С.О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Мулява Г.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Охлопков А.И.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Храмцова С.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Цыплаков С.Р.

Руководитель:  
старший преподаватель департамента ПИиИИ,   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Иваненко Ю.С.

г. Владивосток  
2024

**Оглавление**

[Введение 3](#_Toc124315234)

[1. Разработка плана проекта 4](#_Toc124315235)

[2. Разработка регламента проведения инспекции 5](#_Toc124315236)

[3. Разработка модели состояний задач 9](#_Toc124315237)

[4. Разработка презентации проекта 10](#_Toc124315238)

[5. Разработка требований к проекту 13](#_Toc124315239)

[6. Разработка архитектуры проекта 14](#_Toc124315240)

[7. Разработка измерений проекта 18](#_Toc124315241)

[8. Разработка перечня задач проекта 20](#_Toc124315242)

[9. Разработка рекомендаций по кодированию 22](#_Toc124315243)

[10. Разработка плана тестирования проекта 25](#_Toc124315244)

[11. Тестирование проекта 29](#_Toc124315245)

[Заключение 34](#_Toc124315246)

[Список литературы 35](#_Toc124315247)

# Введение

Промышленная разработка информационных систем включает в себя множество этапов, начиная от разработки плана проекта, заканчивая тестированием проекта для чего, очевидно необходимо множество специалистов различных профилей, а также унифицированные методы коммуникации между ними, с помощью которых можно разделить обязанности членов команды по их специализации.

Исходя из описанного выше необходимо использовать определенные технологии коллективной разработки для повышения эффективности работы в группе и соответствия конечного продукта заявленным требованиям.

В данной курсовой работе рассматривается задача коллективной разработки проекта «Программное средство для навигации в акватории на основе кластеризации траекторий движения судов» и составление технической документации к нему.

Таким образом, целью курсовой работы является разработка разработки проекта «Программное средство для навигации в акватории на основе кластеризации траекторий движения судов» с использованием подходов коллективной промышленной разработки.

Для достижения поставленной цели необходимо решить следующие задачи:

* разработать план проекта;
* разработать регламент проведения инспекции;
* разработать модель состояний задач;
* разработать презентацию проекта;
* разработать требования к проекту;
* разработать архитектуру проекта;
* разработать измерения проекта;
* разработать перечь задач проекта;
* разработать рекомендации по кодированию;
* разработать план тестирования проекта;
* протестировать проект.

# 1. Разработка плана проекта

План проекта – это документ, содержащий подробную информацию о проекте: исполнителях, задачах и сроках. Документ является конечным результатом этапа планирования, утверждается до начала любых работ и становится самым главным и достоверным источником информации о грядущем проекте.

В нашем случае исполнителями являются следующие лица:

* Team Leader - Карпачев Иван
* Backend-разработчик 1 - Цыплаков Станислав
* Backend-разработчик 2 - Михайлов Сергей
* Frontend-разработчик 1 - Храмцова Софья
* Frontend-разработчик 2 - Охлопков Айсен
* Разработчик парсера 1 - Мулява Григорий
* Разработчик парсера 2 - Карпачев Иван

На рисунке ниже представлен перечень задач для выполнения и примерные сроки их реализации:

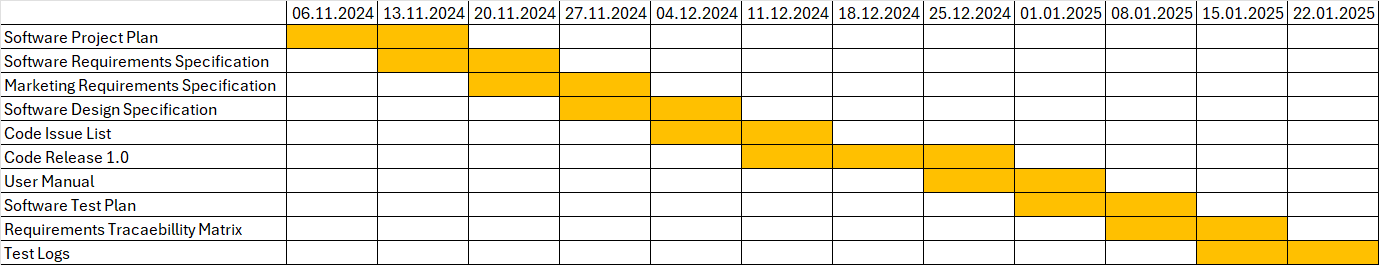


Рисунок 1 – План проекта

# 2. Разработка регламента проведения инспекции

Верификация рабочих продуктов является неотъемлемой частью процесса по обеспечению их качества. Современной технологией программирования выработаны специальные стандарты, подходы и механизмы проведения верификаций рабочих продуктов в формате так называемых инспекций (peer reviews).

Инспекция – это мероприятие по обеспечению качества рабочих продуктов проектов по разработке ПО и иной деятельности, которая проводится разработчиками, возможно - с участием представителей заказчика. Концептуально инспекция имеет следующие цели:

* обнаружить ошибки в функциях, логике, содержании или реализации рабочих продуктов на ранних этапах их разработки и предотвратить их наследование;
* рационально донести замысел или реализацию продукта до всех заинтересованных лиц (через их участие);
* оптимизировать, оценить или улучшить рабочий продукт.

## Критерии отнесения к формальной инспекции

* Разработана новая версия документации или исправлена существующая документация.
* Создание pull request в master в репозиторий на платформе GitHub.

## Перечень ролей участников инспекции и их обязанности

|  |  |
| --- | --- |
| **Роль** | **Обязанности** |
| Автор (Coder 1) | Участник, сделавший инспектируемые изменения в существующем  рабочем продукте |
| Инспектор, проверяющий (Coder 2) | Участник, ответственный за эффективную проверку инспектируемого рабочего продукта. |

Так как в команде два человека, то роли не фиксированы и менялись в зависимости от того, кто внес изменения в проект (тот выступал в роли автора, соответственно).

## Этапы инспекции:

|  |  |
| --- | --- |
| **Этап** | **Что происходит** |
| Назначение инспекции | Автор инициирует формальную инспекцию продукта, уведомляет об этом инспектора |
| Подготовка к инспекции | Инспектор самостоятельно изучает предоставленный для инспекции рабочий продукт. Составляет протокол найденных ошибок. Отправляет его автору |
| Собрания по инспекции | Инспектор самостоятельно изучает предоставленный для инспекции рабочий продукт. Составляет протокол найденных ошибок. Отправляет его автору. |
| Завершение инспекции | Автор анализирует все замечания и исправляет все недостатки рабочего продукта по протоколу инспектора. Отправляет отчет инспектору о том, что все замечания учтены. |

## Порядок организации

Автор выгружает код продукта на гит и приглашает Инспектора. Далее Инспектор готовиться к инспекции и составляет протокол ошибок, которые отправляет автору. После исправления ошибок автор высылает исправленную версию инспектору\проверяющему.

## Порядок подготовки и проведения

Автор уведомляет инспектора в день назначения инспекции. На подготовку инспектора отводится 3-7 дней, в зависимости от сложности и объема инспектируемого продукта. В день завершения подготовки проводится собрание по инспекции, после которого дается 7 дней на исправление ошибок автору. Далее 2 дня на проверку инспектором\проверяющим.

## Перечень статусов и степени важности замечаний

|  |  |
| --- | --- |
| **Перечень статусов** | **Степень важности замечаний** |
| * Дефект (Defect) - проблема, которая найдена на фазе, отличной от той, на которой внесена. * Ошибка (Error) – проблема, которая найдена на той же фазе, на которой внесена. * Комментарий (Comment) – это наблюдение, предложение, рекомендация или улучшение, предложенное для будущего выпуска рабочего продукта или вопрос, требующий разъяснения. Внесение изменения в рабочий продукт в соответствии с комментарием – это результат договоренности автора рабочего продукта и автора комментария. * Замечание для исследования (Investigate) – проблема, природа которой не может быть определена на собрании и требует дополнительного исследования. | * Критическая (Critical) – программа не исполняется * Особо важная (Major) – программа исполняется с ошибкой * Средняя (Moderate) – программа исполняется с временными ошибками при определенных условиях * Мелкая (Minor) – нет защиты от ввода\вывода некорректных данных * Другие (Other)- предложение по оптимизации программы |

## Порядок верификации учёта замечаний

После проведения собрания по инспекции автор за отведенной время исправляет замечания, помечая все исправления в протоколе. Далее этот протокол отправляется проверяющему (инспектору), который проверит, что все замечания учтены в обновленном рабочем продукте корректно. Именно после вердикта проверяющего формальная инспекция считается завершённой.

## Метрики, характеризующие эффективность инспекций

**Inspection Fault Density (IFD)**

IFD характеризует эффективность инспекции, а также качество инспектируемого продукта:

Чем больше IFD, тем эффективнее инспекция при неизменном качестве рабочего продукта и наоборот.

* Стратегическая цель метрики – повысить качество разрабатываемого ПО.
* Изучаемый объект метрики – инспекция.
* Измеряемый атрибут – плотность найденных в ходе инспекции ошибок.
* Единица измерения – ошибка / <страница, требование, LOC, тест>
* Целью предприятия является снижение IFD, чем меньше IFD, тем лучше.

# 3. Разработка модели состояний задач

Каждая задача, являясь отражением делового процесса, проходит определенные состояния. Сначала идет создание задачи, потом идет выполнение работ по задаче, после выполнения задача завершается.

## Перечень состояний задач:

* New – чтобы исправить ошибку или добавить новую функцию, создается задача (потенциальную задачу ставит кодер1 или кодер2);
* On discussion – следующее после состояния New, когда задача обсуждается группой;
* Аpproved – после обсуждения для задачи назначен исполнитель;
* In the process – после выставления статуса Аpproved, исполнитель принял задачу и начал работу над ее реализацией;
* Completed – задача выполнена исполнителем, назначен неформальный инспектор;
* Under review – проводится неформальная инспекция, в случае проблем с проверкой, инспектор назначает эту задачу обратно исполнителю и обратно до тех пор, пока инспекция не пройдет удачно;
* Resolved – назначается в случае удачной инспекции, назначается исполнитель к финальной интеграции в готовый продукт (передается отвечающему за гит);
* Integrated – заинтегрировано (устанавливает участник, ответственный за гит).
* Disapproved – задача на обсуждении признана избыточной, в таком формате ее реализовывать не нужно.

# 4. Разработка презентации проекта

На рисунке 2 представлена титульная страница презентации.



Рисунок 2 – Титульная страница

На рисунке 3 описаны участники команды.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3 – Участники команды.

На рисунке 4 и 5 кратко описана информация о проекте.

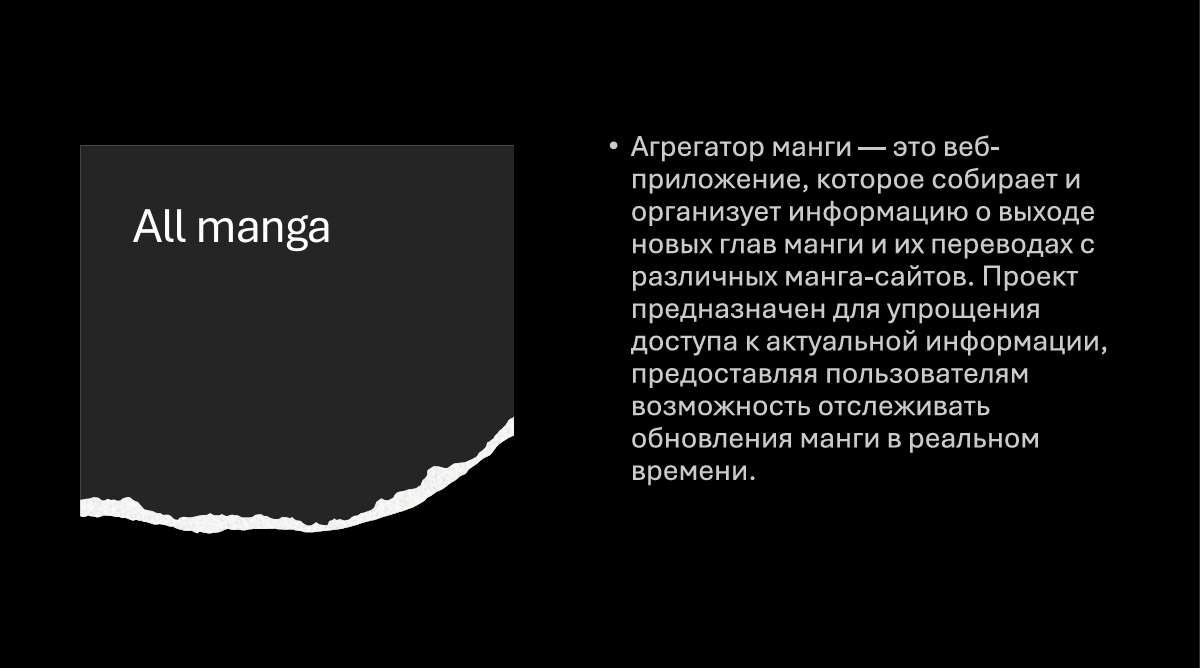


Рисунок 4 – Агрегатор манги.

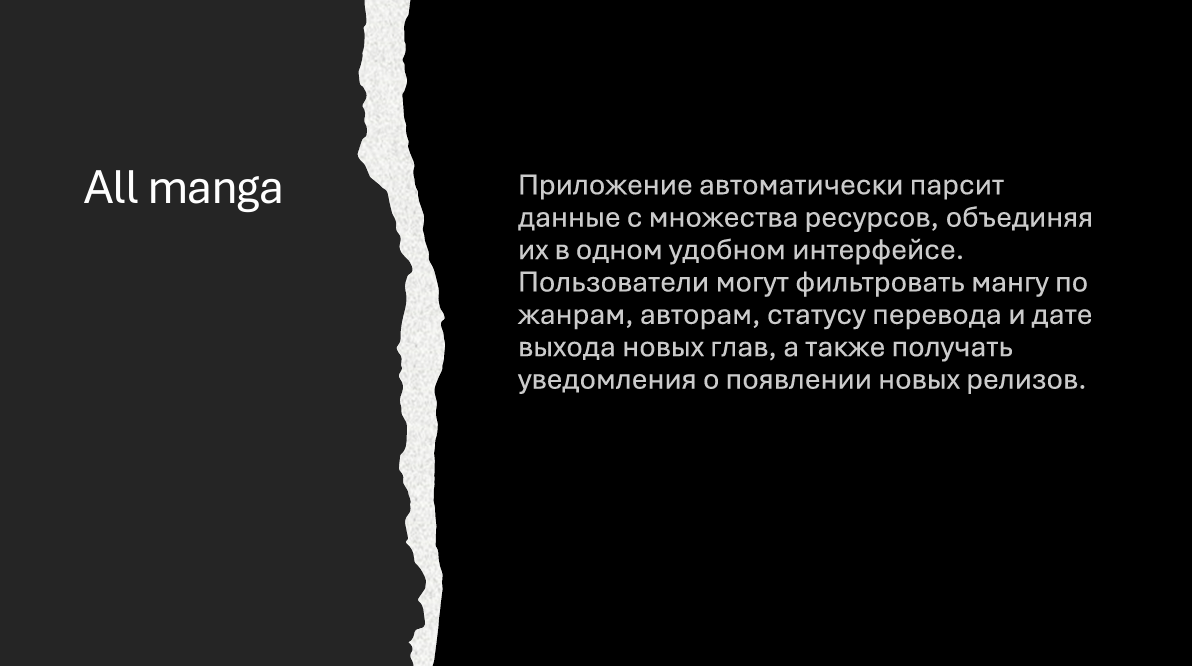


Рисунок 5 – Принцип работы приложения

Преимущества приложения описаны на рисунке 6.

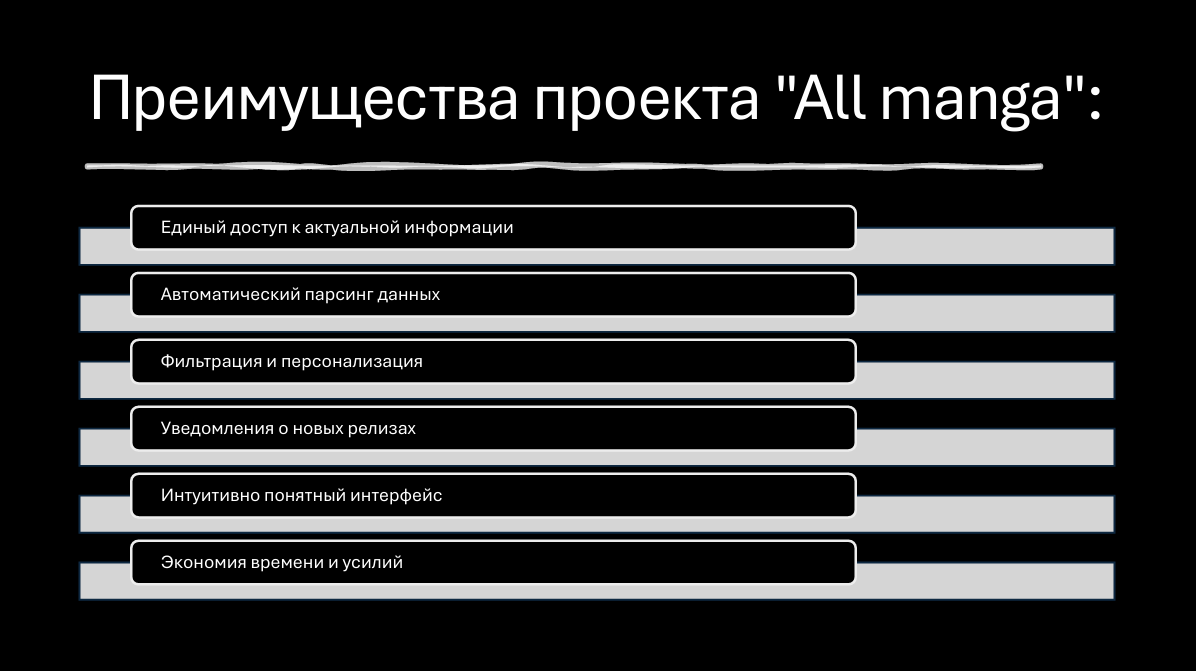


Рисунок 6 – Преимущества проекта.

Основные функции приложения описаны на рисунке 7.

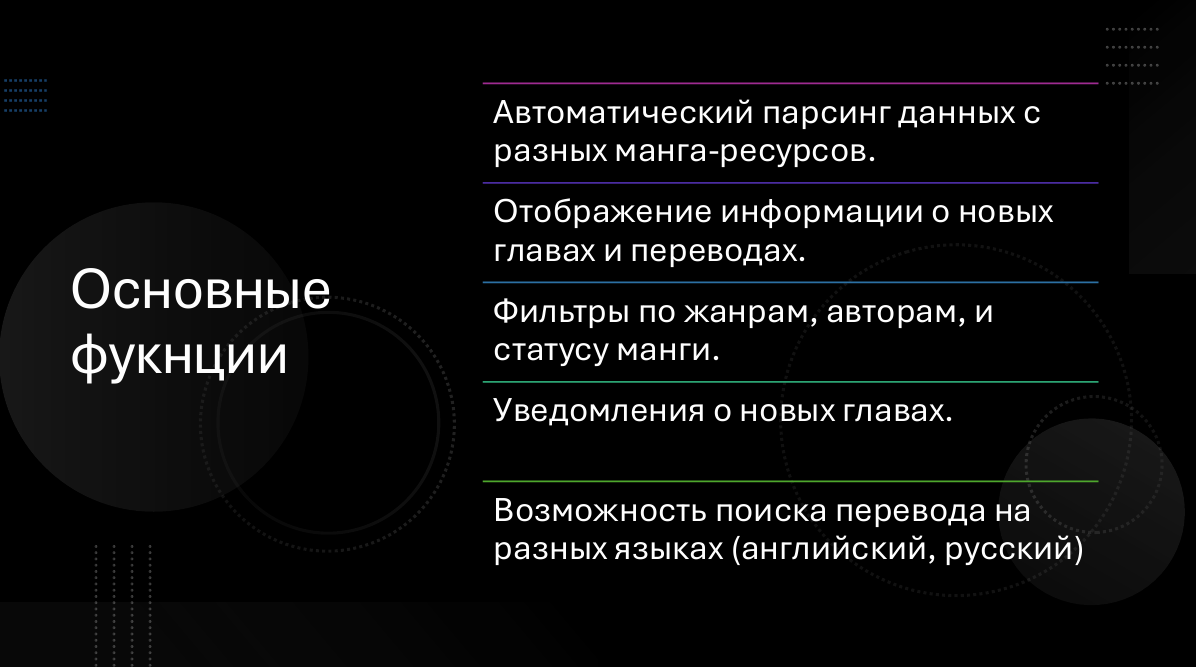


Рисунок 7 – Основные функции.

# 5. Разработка требований к проекту

Программный продукт «All Manga» состоит из следующих подсистем:

* Веб-интерфейс
* Серверная часть
* Парсер

**Требования к подсистеме "Веб-интерфейс":**

1. Система должна предоставлять список всех графических романов
2. Система может отсортировать по:
   * году выхода
   * суммарному рейтингу, взятому с разных источников
   * популярности
3. Система может отфильтровать по:
   * жанру и тегам (Жанры: Исекай, Драма, Сёнен, Триллер, Ужасы и т.д. / Теги: Антигерой, Апокалипсис, Боги, ГГ мужчина, Гоблины и т.д.)
   * возрастному рейтингу (6+, 12+, 16+, 18+)
   * типу произведения (манга, манхва, маньхуа и т.д.)
   * статусу произведения (завершено/выходит/переводится/заброшено)
4. Система может предоставить подробную информацию по каждому произведению:
   * название (в том числе на английском и на языке оригинала)
   * краткое описание
   * автор
   * год выхода
   * жанр и теги
   * рейтинг с разных источников
   * статус произведения (завершено/выходит/переводится/заброшено)
5. Система должна предоставлять статус перевода произведения на различных сервисах для чтения графических романов, посмотреть автора перевода и перейти по URL адресу для последующего чтения

**Требования к подсистеме "Серверная часть":**

1. Система должна хранить информации по каждому произведению в базе данных:
   * название (в том числе на английском и на языке оригинала)
   * краткое описание
   * автор
   * год выхода
   * жанр и теги
   * рейтинг с разных источников
   * статус произведения (завершено/выходить/переводится/заброшено)
   * список глав и статус их перевода, а также ссылку на сайт для чтения главы
2. Система может добавлять, удалять, редактировать информацию по каждому произведению
3. Система должна хранить список сайтов-источников для последующей загрузки в парсер
4. Система может запускать парсер

**Требование к подсистеме "парсер":**

1. Система должна собирать информацию по произвольному произведению с различных источников, список которых хранится на "серверной части"

**Пользовательские требования:**

1. Пользователь может посмотреть список всех произведений;
2. Пользователь может отсортировать список произведений по году выхода, рейтингу и популярности;
3. Пользователь может отфильтровать список произведений по жанру, тегам, возрастному рейтингу, типу произведения и статусу произведения;
4. Пользователь может посмотреть подробную информацию (название, описание, автор, жанры, теги, рейтинг произведения на различных сайтах, статус произведения, информацию о выходе перевода на различных ресурсах) конкретного произведения;
5. Пользователь может перейти по URL адресу на сервисы для чтения выбранного произведения.

# 6. Разработка архитектуры проекта

Архитектура программного обеспечения относится к фундаментальным структурам программной системы и дисциплине создания таких структур и систем. Каждая структура включает элементы программного обеспечения, отношения между ними, а также свойства как элементов, так и отношений. Архитектура программной системы – это метафора, аналогичная архитектуре здания. Он функционирует как план для системы и проекта разработки, в котором излагаются задачи, которые должны быть выполнены командами разработчиков.

Одним из способов представления архитектуры проекта является модель С4. Она состоит из четырех диаграмм: Context diagrams (level 1), Container diagrams (level 2), Component diagrams (level 3), Code diagrams (level 4).

Модель C4 - достаточно новая и простая архитектурная модель для проектирования системы. Работает по принципу масштабирования картинки. То есть, увеличиваем масштаб - увеличиваем детализацию схемы.

Диаграммы С4 представляют собой иерархию функциональных процессов, связанных потоками данных, различного масштаба. Цель такого представления — продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

На рисунке 8 представлена контекстная диаграмма первого уровня.



Рисунок 8 – Контекстная диаграмма

На рисунке 9 представлен графический сценарий диалога.

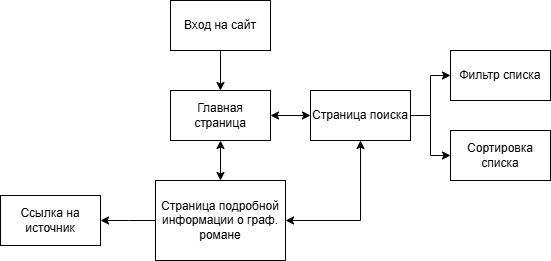


Рисунок 9 – Графический сценарий диалога.

# 7. Разработка измерений проекта

Контроль за производственным процессом и его результатами является ключевым видом деятельности на современном предприятии, производящем программное обеспечение на заказ. В силу специфики такого продукта, как программное обеспечение, для оценки эффективности процесса и качества конечного продукта применяются особые методы. Комплекс мероприятий, направленных на количественную оценку эффективности работы компании, называется программой измерений компании. Программа измерений выполняется как в рамках отдельных проектов, осуществляемых компанией, так и в рамках определённых видов деятельности компании.

## Оценка эффективности процесса разработки

* **Productivity = LOC / Рабочее время, затраченное на проект**

Стратегическая цель метрики – повысить производительность труда;

Изучаемый объект метрики – проект;

Измеряемый атрибут – производительность труда на проекте;

Единица измерения – LOC / человеко-час.

* **On project % time (OPPT) = (Рабочее время, затраченное на проект / Общее рабочее время) \* 100%;**

Стратегическая цель метрики – сокращение сроков выполнения проектов по разработке ПО;

Изучаемый объект метрики – рабочее время;

Измеряемый атрибут – доля рабочего времени, затраченная на работу по проектам;

Единица измерения – %.

## Оценка качества программного продукта

* **Product Fault Density (PFD)**

**PFD-Req** = Число ошибок, допущенных на этапе разработки требований / Размер требований;

**PFD-Design** = Число ошибок, допущенных на этапе дизайна / Размер документов дизайна;

**PFD-Coding** = Число ошибок, допущенных на этапе кодирования / LOC;

**PFD-Test** = Число ошибок, допущенных на этапе тестирования / Количество тестов;

Стратегическая цель метрики – повысить качество разрабатываемого ПО;

Изучаемый объект метрики – продукт;

Измеряемый атрибут – плотность неполадок (учитывает все ошибки);

Единица измерения – неполадка / единица размера.

* **In Process Faults (IPF) = (Число обнаруженных ошибок до выпуска его релиза) / LOC**

Стратегическая цель метрики – повысить качество разрабатываемого ПО;

Изучаемый объект метрики – продукт;

Измеряемый атрибут – плотность неполадок;

Единица измерения – неполадка / LOC.

# 8. Разработка перечня задач проекта

**Задачи для подсистемы «Веб-интерфейс»**

*Задача TASK\_UI\_1*

Название: Проектирование интерфейса главной страницы.

Описание: основываясь на требованиях, разработать интерфейс.

Минимальный функционал: отображение произведений по актуальности, ссылок на подробное описание конкретного произведения, кнопки перехода на страницу поиска.

*Задача TASK\_UI\_2*

Название: Проектирование интерфейса поисковой машины.

Описание: основываясь на требованиях, разработать интерфейс.

Минимальный функционал: отображение поисковой строки, фильтров для поиска, сортировки произведений, результатов поиска.

*Задача TASK\_UI\_3*

Название: Проектирование интерфейса страницы информации о произведении.

Описание: основываясь на требованиях, разработать интерфейс.

Минимальный функционал: отображение информации о произведении и ссылок на сайты для чтения этой работы.

**Задачи для подсистемы «Серверная часть»**

*Задача TASK\_SP\_1*

Название: Разработка серверной части

Описание: основываясь на требованиях, разработать серверную часть, которая будет в ответ на запрос пользователя отправлять данные из базы данных.

*Задача TASK\_SP\_2*

Название: Подключение базы данных.

Описание: основываясь на требованиях, подключить базу данных для хранения информации о произведениях.

**Задачи для подсистемы «Парсер»**

*Задача TASK\_*PA*\_1*

Название: Разработка парсера произведений.

Описание: основываясь на требованиях, разработать парсер произведений.

Минимальный функционал: периодически подгружать информацию, собранную с сайтов произведений, в базу данных.

**Задачи для системы в целом**

*Задача TASK\_SYS\_1*

Название: Сборка Веб-сервиса

Описание: собрать готовый Веб-сервис по поиску произведений.

# 9. Разработка рекомендаций по кодированию

Для создания качественного кода на любом языке программирования, обладающего таким свойствами, как удобочитаемость (readability) и понятность (understandability), необходимо следовать хорошо определённым стандартам и руководствам. Особенно это актуально при коллективной разработке программ. Любой стандарт кодирования призван определить набор правил, которые способствуют разработке более единообразного кода и минимизации числа общераспространенных ошибок в нем, не ущемляя при этом права разработчика на творчество.

Программная система будет реализована на языках программирования Python и JavaScript.

PEP 8 — документ, описывающий соглашение о том, как писать код на языке Python. PEP 8 создан на основе рекомендаций создателя языка Гвидо ван Россума.  Ключевая идея Гвидо такова: код читается намного больше раз, чем пишется.

Для JavaScript нет единого стандарта написания кода, поэтому в этом проекте будет использоваться рекомендации от Google: Google JavaScript Style Guide.

Собственно, рекомендации о стиле написания кода направлены на то, чтобы улучшить читаемость кода и сделать его согласованным между большим числом проектов. В идеале, если весь код будет написан в едином стиле, то любой сможет легко его прочесть.

## Основные правила PEP 8

* **Форматирование**

 Использовать четыре пробела для отступов. Не делать отступов в два пробела. Wing Ide помогает правильно расставлять пробелы. По умолчанию в Wing Ide клавиша Tab ставит четыре пробела.

Писать import  каждого модуля в отдельной строке

Располагать все import'ы в верхней части кода перед любыми глобальными объявлениями.

Отделять блок import'ов от кода пустой строкой.

Не использовать конструкцию from … import \*}

Скобки не отделяются пробелами с внутренней стороны. Между функцией и ее аргументами пробел не ставится.

Пример:

spam(ham[1], {eggs: 2})         # Правильно  
spam( ham[ 1 ], { eggs: 2 } )  # Неверно

Перед запятой, двоеточием пробел не ставится, после -— ставится.

Пример:

if x == 4:

    print(x, y)

    x, y = y, x       # Правильно

if x == 4 :

    print(x , y)

    x , y = y , x     # Неверно

 Всегда необходимо окружать следующие бинарные операторы ровно одним символом пробела с каждой стороны:

* присваивания (=, +=, -= и т. д.),
* сравнения (==, <, >, !=, <>, <=, >=, in, not in, is, is not),
* логические (and, or, not),
* арифметические (+, -, \*, /, //, \%, \*\*).

Не располагайте несколько инструкций в одной строке – разносить по разным строкам.

Пример:

x = 3           # Правильно

func(10)

x = 3; func(10)  # Неверно

Не располагать блок из нескольких инструкций на той же строке сразу после двоеточия (после if, while и т. д.)

* **Комментарии**

Комментарии, противоречащие коду, хуже, чем их отсутствие.

Необходимо располагать однострочные комментарии после кода в той же строке и отделяйте их от кода не менее чем двумя пробелами. Комментарии должны начинаться с # и одного пробела.

* **Имена**

Не использовать символы `l', `O', и `I' как имена переменных. В некоторых шрифтах они могут быть очень похожи на цифры.

Имена переменных и функций должны содержать только маленькие буквы. Слова разделяются символами подчёркивания.

Примеры:

 name, name\_with\_several\_words\_in\_it

Имена констант должны содержать только заглавные буквы. Слова разделяются символами подчёркивания.

Примеры:

 NAME, NAME\_WITH\_SEVERAL\_WORDS\_IN\_IT

Необходимо давать переменным говорящие английские имена - не использовать транслит.

num\_letters = int(input())         # Правильно

kolvo\_bukv = int(input())          # Неверно

* **Функции**

Необходимо разделять определения функций двумя пустыми строками, а также отделять определения функций от основного кода двумя пустыми строками до и после.

Тело функции разделять на логические части одной пустой строкой.

## Основные правила Google JavaScript Style Guide

**1. Форматирование**

1. **Отступы:**
   * Используйте **два пробела** для отступов, вместо табуляций.
   * Не используйте смешанные типы отступов (табуляции + пробелы).
2. **Импорты:**
   * Располагайте все import в верхней части файла перед любыми другими инструкциями.
   * Каждый модуль импортируйте в отдельной строке.
   * Используйте группировку импортов:
     + Стандартные модули (например, fs, path).
     + Библиотеки (например, lodash, axios).
     + Локальные модули.
   * Не используйте конструкцию import \*.

**Пример:**

import fs from 'fs'

import path from 'path'

import \_ from 'lodash'

import myModule from './myModule.js'

1. **Скобки и пробелы:**
   * Внутренние стороны скобок не отделяются пробелами.
   * Между функцией и её аргументами пробел не ставится.
   * Скобки всегда используются в многострочных условиях или функциях, даже если они необязательны.

**Пример:**

// Правильно

someFunction(arg1, { key: value })

if (x === 10) {

console.log('Value is 10')

}

// Неправильно

someFunction( arg1, { key: value } )

if (x === 10) console.log('Value is 10')

1. **Пробелы:**
   * Пробел не ставится перед запятой, но всегда ставится после.
   * Пробел ставится вокруг бинарных операторов: =, +, -, <, >, и т. д.

**Пример:**

// Правильно

const total = a + b

const values = [1, 2, 3]

// Неправильно

const total=a+b

const values=[1,2,3]

1. **Длина строки:**
   * Максимальная длина строки — 80 символов.
   * Для длинных строк используйте переносы, выравнивая код по логическому блоку.
2. **Многострочные структуры:**
   * Если объект, массив или вызов функции занимает несколько строк, используйте отступы для читаемости.

**Пример:**

const config = {

host: 'localhost',

port: 8080,

credentials: {

user: 'admin',

pass: 'password'

}

}

**2. Комментарии**

1. **Общие правила:**
   * Комментарии должны быть на английском языке.
   * Комментарии, противоречащие коду, хуже их отсутствия.
   * Используйте комментарии для пояснения сложной логики, но избегайте излишнего документирования очевидного.
2. **Однострочные комментарии:**
   * Используйте //.
   * Располагаются в отдельной строке над блоком кода или в той же строке, что и код, с двумя пробелами перед началом комментария.

**Пример:**

// инициализация переменных

let total = 0

const maxLimit = 100 // максимально допустимое значение

1. **Многострочные комментарии:**
   * Используйте /\* ... \*/ для документирования функций, классов и сложных блоков кода.

**Пример:**

/\*\*

\*Вычисление квадрата числа.

\* @param {number} num – число для возведения в квадрат.

\* @return {number}квадрат введенного числа.

\*/

function square(num) {

return num \* num

}

**3. Имена**

1. **Переменные и функции:**
   * Используйте camelCase для переменных и функций.
   * Не используйте символы, которые сложно различить (l, I, O).
   * Имена переменных должны быть говорящими и описывать цель использования.

**Пример:**

let userAge = 25

function calculateDiscount(price) {

return price \* 0.1

}

1. **Константы:**
   * Константы пишутся в UPPER\_SNAKE\_CASE.
   * Пример:

const MAX\_LIMIT = 100

const API\_URL = 'https://api.example.com'

1. **Классы и перечисления:**
   * Используйте PascalCase.
   * Пример:

class UserProfile {

constructor(name, age) {

this.name = name

this.age = age

}

}

**4. Функции**

1. **Структура функций:**
   * Разделяйте логические блоки кода пустой строкой.
   * Используйте стрелочные функции там, где это уместно.

**Пример:**

const greet = (name) => {

if (!name) {

return 'Hello, Guest!'

}

return `Hello, ${name}!`

}

1. **Разделение определений:**
   * Отделяйте определения функций и классов двумя пустыми строками.

**Пример:**

function add(a, b) {

return a + b

}

function subtract(a, b) {

return a - b

}

**5. Прочие рекомендации**

1. **Не пишите несколько инструкций в одной строке.**

// Правильно

let x = 10

console.log(x)

// Неправильно

let x = 10; console.log(x)

1. **Используйте строгий режим (use strict) везде, где это возможно.**

'use strict'

1. **Ограничивайте вложенность.**
   * Глубокая вложенность делает код сложным для понимания. Разделяйте блоки кода на функции.

**Пример:**

// Плохо

if (condition1) {

if (condition2) {

if (condition3) {

doSomething()

}

}

}

// Хорошо

if (condition1 && condition2 && condition3) {

doSomething()

}

**Пример кода с примененными рекомендациями:**

'use strict'

// Импорт модулей

import fs from 'fs'

import path from 'path'

// Константы

const MAX\_LIMIT = 100

// Функция для вычисления квдарата

/\*\*

\*Вычисляет квадрат числа.

\* @param {number} num -введенное число.

\* @return {number}Квадрат введенного числа.

\*/

function square(num) {

return num \* num

}

const result = square(10)

console.log(result)

# 10. Разработка плана тестирования проекта

## Тесты для тестирования подсистемы «Веб-интерфейс»

***Тест TEST\_UI\_1***

Тестируемые требования: *REQ\_UI\_1*.

Описание теста: вход на главную страницу сайта.

Ожидаемый результат: страница со списком всех графических романов.

***Тест TEST\_UI\_2***

Тестируемые требования: *REQ\_UI\_2*.

Описание теста: отсортировать список графических романов.

Ожидаемый результат: отсортированный список графических романов.

***Тест TEST\_UI\_3***

Тестируемые требования: *REQ\_UI\_3*.

Описание теста: отфильтровать список графических романов.

Ожидаемый результат: отфильтрованный список графических романов.

***Тест TEST\_UI\_4***

Тестируемые требования: *REQ\_UI\_4*.

Описание теста: переход на страницу конкретного графического романа.

Ожидаемый результат: страница графического романа с подробной информацией о нём.

***Тест TEST\_UI\_5***

Тестируемые требования: *REQ\_UI\_5*.

Описание теста: переход на страницу конкретного графического романа.

Ожидаемый результат: страница графического романа с информацией о статусе перевода, авторе перевода и URL адресом на сайт с переводом.

## Тесты для тестирования подсистемы «Серверная часть»

***Тест TEST\_SP\_1***

Тестируемые требования: *REQ\_SP\_1, REQ\_SP\_2.*

Описание теста: добавление в базу данных объекта графического романа.

Ожидаемый результат: в базе данных появится добавляемый объект.

***Тест TEST\_SP\_2***

Тестируемые требования: *REQ\_SP\_2.*

Описание теста: удаление объекта из базы данных.

Ожидаемый результат: в базе данных исчезнет удаляемый объект.

***Тест TEST\_SP\_3***

Тестируемые требования: *REQ\_SP\_2.*

Описание теста: редактирование объекта в базе данных.

Ожидаемый результат: редактируемый объект в базе данных изменится.

***Тест TEST\_SP\_4***

Тестируемые требования: *REQ\_SP\_3.*

Описание теста: получение списка сайтов-источников.

Ожидаемый результат: список сайтов-источников.

***Тест TEST\_SP\_5***

Тестируемые требования: *REQ\_SP\_4.*

Описание теста: запуск парсера.

Ожидаемый результат: запущенный парсер.

## Тесты для тестирования подсистемы «Парсер»

***Тест TEST\_PA\_1***

Тестируемые требования: *REQ\_PA\_1.*

Описание теста: запуск программы

Ожидаемый результат: собранная информация, находящаяся в базе данных на серверной части.

## Тесты для тестирования системы в целом

***Тест TEST\_SYS\_1***

Тестируемые требования: *REQ\_UI\_1, REQ\_UI\_2, REQ\_UI\_3, REQ\_UI\_4, REQ\_UI\_5, REQ\_SP\_1, REQ\_SP\_2, REQ\_SP\_3, REQ\_SP\_4, REQ\_PA\_1.*

Описание теста: запустить серверную часть, запустить парсер, использовать сайт по назначению.

Ожидаемый результат: система состоящая из парсера, который собирает информацию по источникам и отправляет её на серверную часть в базу данных, серверной части, которая обрабатывает запросы пользователей, запускает парсер и совершает работу с базой данных и веб-интерфейса, в котором возможно получить список графических романов, можно их отфильтровать, отсортировать, посмотреть подробную информацию о каждом.

## Матрица покрытия тестами требований

Матрица соответствия требований используется QA-инженерами для валидации покрытия требований по продукту тестами. Цель «Traceability Matrix» состоит в том, чтобы выяснить:

* какие требования «покрыты» тестами, а какие нет;
* избыточность тестов (одно функциональное требование покрыто большим количеством тестов).

Данный тестовый артефакт является неотъемлемой частью тестирования.

В соответствии с написанными требованиями и тестами на Рисунке 10 представлена матрица покрытия тестами требований.

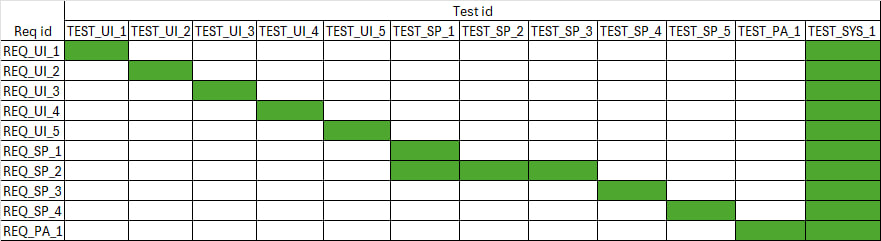


Рисунок 10 - Матрица покрытия тестами требований

# 11. Тестирование проекта

## Тесты для тестирования подсистемы «Пользовательский интерфейс»

***Тест TEST\_UI\_001***

Тестируемые требования: *REQ\_UI\_001*.

Описание теста: запустить выполнение кода.

Ожидаемый результат: при запуске программы появляется окошка интерфейса, содержащее картинку карты хорошего разрешения и кнопки для отображения точек-кластеров-полигонов на основе подгружаемых данных о маршрутах акватории.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

## Тесты для тестирования подсистемы «Модуль загрузки и обработки данных»

***Тест TEST\_LD\_001***

Тестируемые требования: *REQ\_LD\_001, REQ\_UI\_001, REQ\_UI\_002.*

Описание теста: при загрузки данных должна происходить их обработка таким образом, что из загружаемых данных необходимо удалить повторяющиеся строки, удалить данные о судах с курсом 511, а также удалить информацию о судах со значением 0 в столбцах length и port – это могут быть статичные объекты, например буйки.

Ожидаемый результат: количество записей уменьшится в несколько раз, а точность выполнение алгоритма на их основе возрастет или не изменится.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

## Тесты для тестирования подсистемы «Модуль вычислений»

***Тест TEST\_DM\_001***

Тестируемые требования: *REQ\_DM\_001, REQ\_UI\_001, REQ\_UI\_002.*

Описание теста:

1. Задать параметры алгоритма в коде;
2. Нажать выполнение;
3. Визуализировать результаты на карте;
4. Изменить параметры и повторить шаг 1-3;
5. Сравнить результаты.

Ожидаемый результат: при различных параметрах алгоритм будет отдавать разную ситуацию в акватории. При наилучших подобранных параметрах ожидается наиболее точных результат кластеризации и выделения полигонов.

Видимый результат: частично совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: хоть подбор параметров прописывается в коде, но оценить точность результатов работы алгоритма затруднительно.

## Тесты для тестирования подсистемы «Модуль нахождения кратчайшего пути»

***Тест TEST\_FR\_001***

Тестируемые требования: *REQ\_FR\_001, REQ\_UI\_001.*

Описание теста: нажать на кнопку «Выполнить алгоритм Дейкстры».

Ожидаемый результат: на карте отобразится результат алгоритма, картинка карты сохранится в папку img.

Видимый результат: модуль не реализован.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест не пройден.

***Тест TEST\_FR\_002***

Тестируемые требования: *REQ\_FR\_001, REQ\_UI\_001.*

Описание теста: нажать на кнопку «Выполнить алгоритм поиска A\*».

Ожидаемый результат: на карте отобразится результат алгоритма, картинка карты сохранится в папку img.

Видимый результат: модуль не реализован.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест не пройден.

***Тест TEST\_FR\_003***

Тестируемые требования: *REQ\_FR\_001, REQ\_UI\_001.*

Описание теста: нажать на кнопку «Выполнить алгоритм Ли».

Ожидаемый результат: на карте отобразится результат алгоритма, картинка карты сохранится в папку img.

Видимый результат: модуль не реализован.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест не пройден.

## Тесты для тестирования подсистемы «Модуль отображения полученных данных»

***Тест TEST\_SR\_001***

Тестируемые требования: *REQ\_SR\_001,* *REQ\_UI\_001, REQ\_LD\_001, REQ\_DM\_001*.

Описание теста: запустить выполнение кода.

Ожидаемый результат: на основе данных маршрутов акватории, создастся скриншот карты (требуется интернет-соединение для подключения к сайту, откуда необходимо отскринить изображение), с которого обрежется картинка, которая сохранится в папку img и отобразится в окошке интерфейса.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

***Тест TEST\_SR\_002***

Тестируемые требования: *REQ\_SR\_001, REQ\_UI\_001, REQ\_UI\_002, REQ\_LD\_001, REQ\_DM\_001.*

Описание теста: нажать на кнопку «Отобразить все данные».

Ожидаемый результат: на карте отобразятся все записи из файла в формате точек-кластеров-полигонов, измененная картинка сохранится в папку img.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

***Тест TEST\_SR\_003***

Тестируемые требования: *REQ\_SR\_001,* *REQ\_UI\_002, REQ\_LD\_00, REQ\_DM\_0011.*

Описание теста: нажать на кнопку «Отобразить первые 100 записей».

Ожидаемый результат: на карте отобразится часть записей из файла в формате точек-кластеров-полигонов, измененная картинка сохранится в папку img.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

## Тесты для тестирования системы в целом

***Тест TEST\_SYS\_001***

Тестируемые требования: *REQ\_UI\_001, REQ\_UI\_002, REQ\_LD\_001, REQ\_DM\_001, REQ\_SR\_001.*

Описание теста: запустить программу, нажать кнопку «Показать все записи».

Ожидаемый результат: запуститься окно интерфейса, на основе коллекции координат суден погрузится изображении части карты. После нажатия на кнопку, на карте отобразятся точки-кластеры-полигоны, полученные из предобработанных данных о маршрутах акватории.

Видимый результат: полностью совпадает с ожидаемым.

Тестируемая версия продукта: 1.0.0.

Резюме: Тест пройден.

# Заключение

В рамках курсовой работы было разработано программное средство «Программная система для навигации в акватории на основе кластеризации траекторий движения судов» с использованием подходов коллективной промышленной разработки, для чего были решены следующие поставленные задачи:

* разработан план проекта;
* разработан регламент проведения инспекции;
* разработана модель состояний задач;
* разработана презентацию проекта;
* разработаны требования к проекту;
* разработана архитектуру проекта;
* разработаны измерения проекта;
* разработан перечь задач проекта;
* разработаны рекомендации по кодированию;
* разработан план тестирования проекта;
* протестирован проект.

По окончании всех работ посчитаем метрики, описанные в 7 главе.

* **Productivity** = 300 / 10 = 30
* **Problem Resolution Rate** = 1
* **Product Fault Density** = 3.5 / 11 = 30%

Таким образом, цель данной курсовой работы была достигнута.

# Список литературы

1. Гриняк В.М. Лекции по дисциплине «Технологии коллективной промышленной разработки информационных систем». Электронный вариант.