  
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**

**РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА ДЛЯ ОБУСТРОЙСТВА ИНТЕРЬЕРА С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИЙ КОЛЛЕКТИВНОЙ ПРОМЫШЛЕННОЙ РАЗРАБОТКИ**

КУРСОВОЙ ПРОЕКТ  
по дисциплине «Технологии коллективной промышленной разработки информационных систем» по образовательной программе подготовки бакалавров по направлению 02.03.03 «Технология программирования»

Выполнили:  
студенты гр. Б9121-02.03.02тп  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Мылахова С.А.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Григорьев А.А.  
Руководитель:  
старший преподаватель департамента ПИиИИ  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Иваненко Ю.С.

г. Владивосток  
2025

**Оглавление**

[Введение 3](#_Toc188205354)

[1 План проекта 4](#_Toc188205355)

[2 Регламент проведения инспекции 5](#_Toc188205356)

[3 Модель состояний задач 8](#_Toc188205357)

[4 Презентация проекта 10](#_Toc188205358)

[5 Требований к проекту 13](#_Toc188205359)

[6 Разработка архитектуры проекта 17](#_Toc188205360)

[7 Измерения проекта 21](#_Toc188205361)

[Метрики эффективности производственного процесса 21](#_Toc188205362)

[Метрики качества продукта 21](#_Toc188205363)

# Введение

Промышленная разработка информационных систем включает в себя множество этапов, начиная от разработки плана проекта, заканчивая его тестированием, для чего 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 программного средства «Программная система для автоматического создания выкроек» и составление технической документации к нему.

Таким образом, целью курсовой работы является разработка программного средства с использованием подходов коллективной промышленной разработки.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать план проекта.
2. Разработать регламент проведения инспекции.
3. Разработать модель состояний задач.
4. Разработать презентацию проекта.
5. Разработать требования к проекту.
6. Разработать архитектуру проекта.
7. Разработать измерения проекта.
8. Разработать перечень задач проекта.
9. Разработать рекомендации по кодированию.
10. Разработать план тестирования проекта.
11. Протестировать проект.

# 1 План 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:

Coder 1– Григорьев Алексей

Coder 2 – Мылахова Светлана

На рисунке 1 представлен перечень задач для выполнения и примерные сроки их реализации

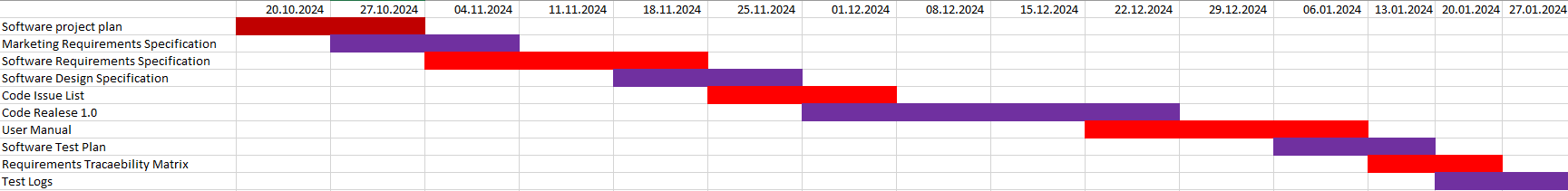


Рисунок 1 – План проекта

# 2 Регламент 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.

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.

Перечень ролей участников инспекции и их обязанности

Таблица 1. Роли и обязанности участников разработки и проверки кода

|  |  |
| --- | --- |
| **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. |

Этапы инспекции

Планирование инспекции.

При планировании инспекции коллективно выбирается дата, время, формат (очный или заочный) и платформа (при заочной инспекции) проведения инспекции.

Подготовка к инспекции.

Инспектор самостоятельно изучает предоставленный для инспекции рабочий продукт, используя накопленный опыт и стандарты.

Собрание по инспекции.

На собрании происходит обсуждение замечаний и рекомендаций инспектора по рабочему продукту. На собрании по инспекции обязательно присутствует инспектор и автор рабочего продукта, требующего инспекции. Присутствие остальных участников команды разработки по желанию.

Завершение инспекции.

Если рабочий продукт требует доработки, то автор фиксирует все замечания и рекомендации инспектора, разрабатывает план предстоящих работ и далее согласовывает его с инспектором. Если рабочий продукт не требует доработки, то инспектор подтверждает слияние рабочей ветки в «master» ветку.

Перечень статусов и степени важности замечаний

Ошибка – проблема, которая найдена на той же фазе, на которой внесена. Допустимые значения степени серьёзности ошибки:

Критическая (Critical)

Средняя (Moderate)

Мелкая, незначительная (Minor)

Другие (Other)

Комментарий – это наблюдение, предложение, рекомендация или улучшение, предложенное для будущего выпуска рабочего продукта или вопрос, требующий разъяснения.

Метрики, характеризующие эффективность инспекций

Стратегическая цель метрики – повысить качество разрабатываемого ПО.

В качестве метрики, характеризующей эффективность инспекции, была выбрана Inspection Preparation Rate (IPR):

IPR = (Количество инспекторов \* Размер продукта) / Общее время подготовки

Изучаемый объект метрики – подготовка к инспекции, измеряемый атрибут – производительность подготовки к инспекции.

Единица измерения – <страница, требование, LOC, тест> / час

# 3 Модель 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 – новая подзадача.

Analysis – в процессе анализа. В это состояние подзадачу переводит сотрудник после того, как начнёт её анализ.

Forward – в данном случае имеет значение «переданный на дальнейшую разработку». В это состояние задача переводится CCB после анализа при назначении задачи на разработку конкретному сотруднику.

Coding – кодирование. В это состояние задача переводится сотрудникомразработчиком, при начале работы по кодированию, связанному с задачей.

Inspected – проинспектировано. В это состояние задача переводится сотрудником-разработчиком после завершения кодирования и инспектирования изменений рабочего продукта.

Integrated – заинтегрировано. Переводится сотрудником, осуществляющим интеграцию изменений в основную ветку рабочего продукта после успешной интеграции этих изменений.

Tested – протестировано. Переводится сотрудником, осуществляющим тестирование изменений в рабочий продукт (tester).

Closed – закрыто. В это состояние задача переводится CCB по результатам отчёта о тестировании сделанных изменений.

Правила создания новой задачи

Созданием новых задач могут заниматься все участники команды. Происходит это на начальном этапе, когда необходимо организовать структурированную работу над проектом и в процессе работы над проектом в случае присутствия в плане слишком объемных задач, требующих упрощения. Также, когда необходимо организовать баг-фиксинг или разработать новый функционал.

Правила перехода задачи из состояния в состояние

Состояния задач всегда идут последовательно друг за другом, в некоторых случаях пункты могут опускаться или повторяться.

# 4 Презентация проекта

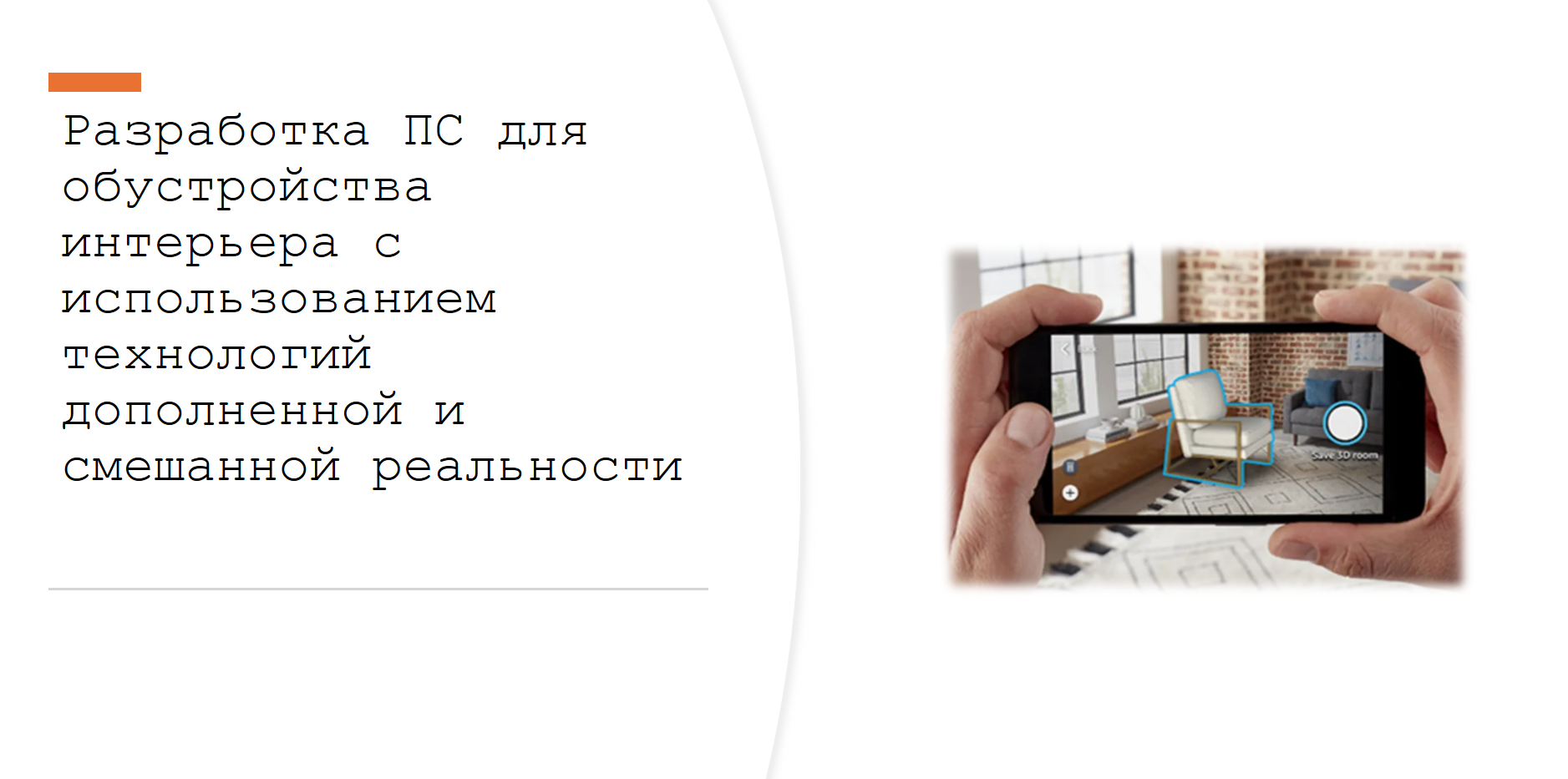


Рисунок 2 – Титульная страница

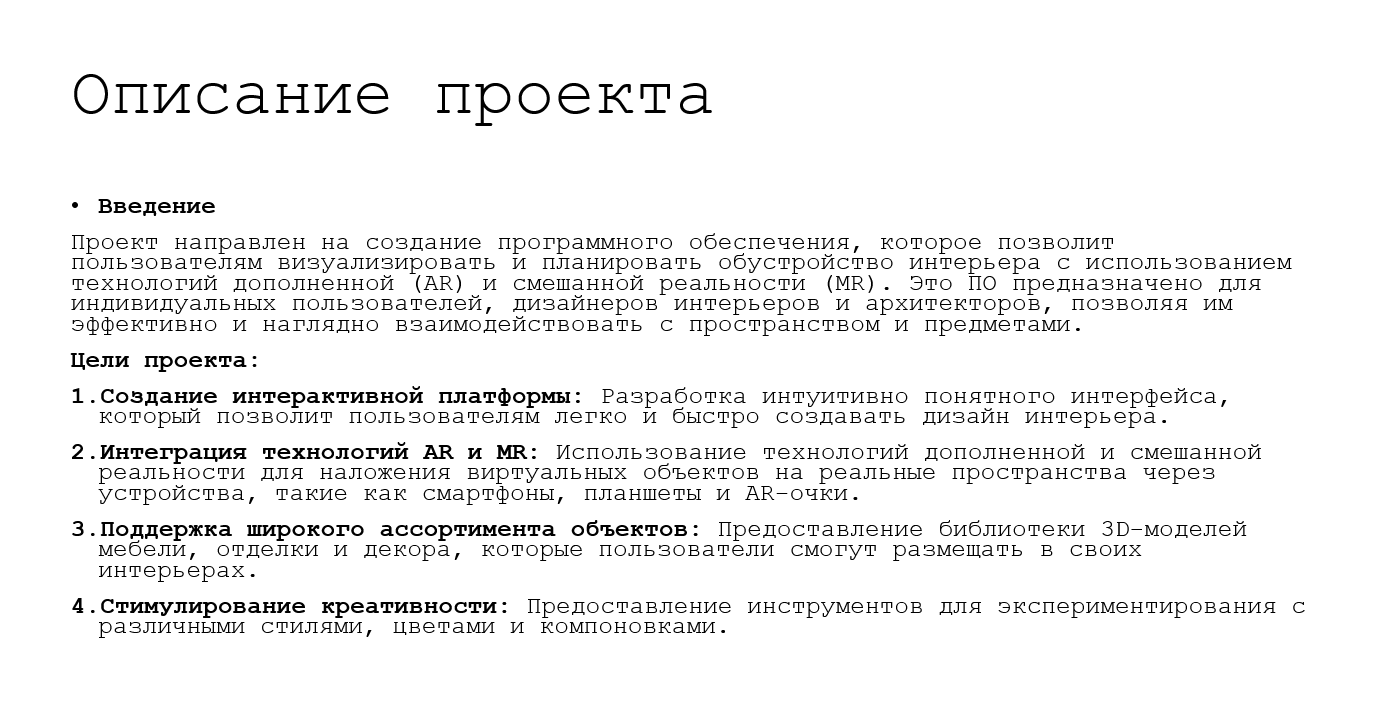


Рисунок 3 – Описание проекта

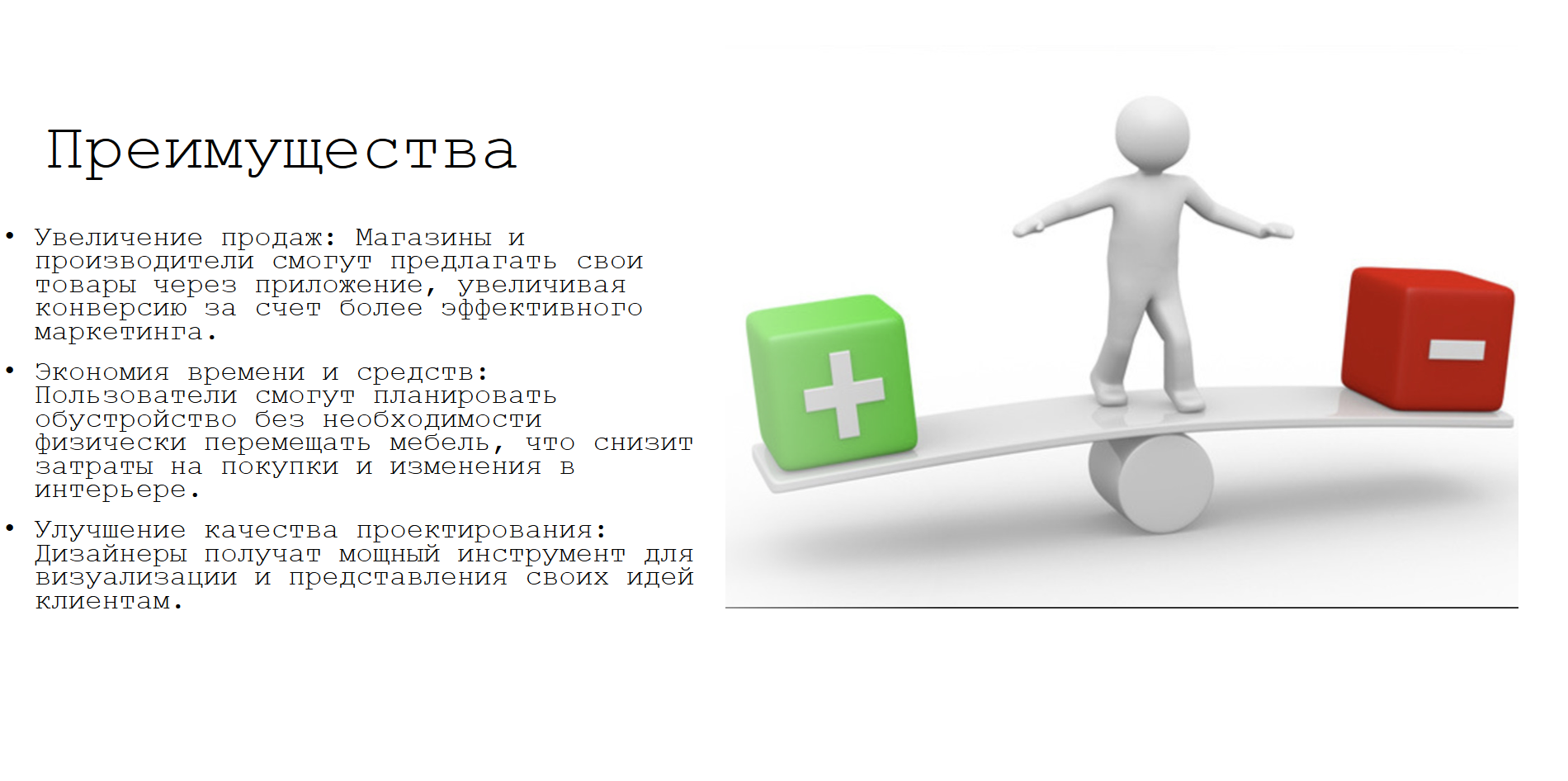


Рисунок 4 – Преимущества проекта

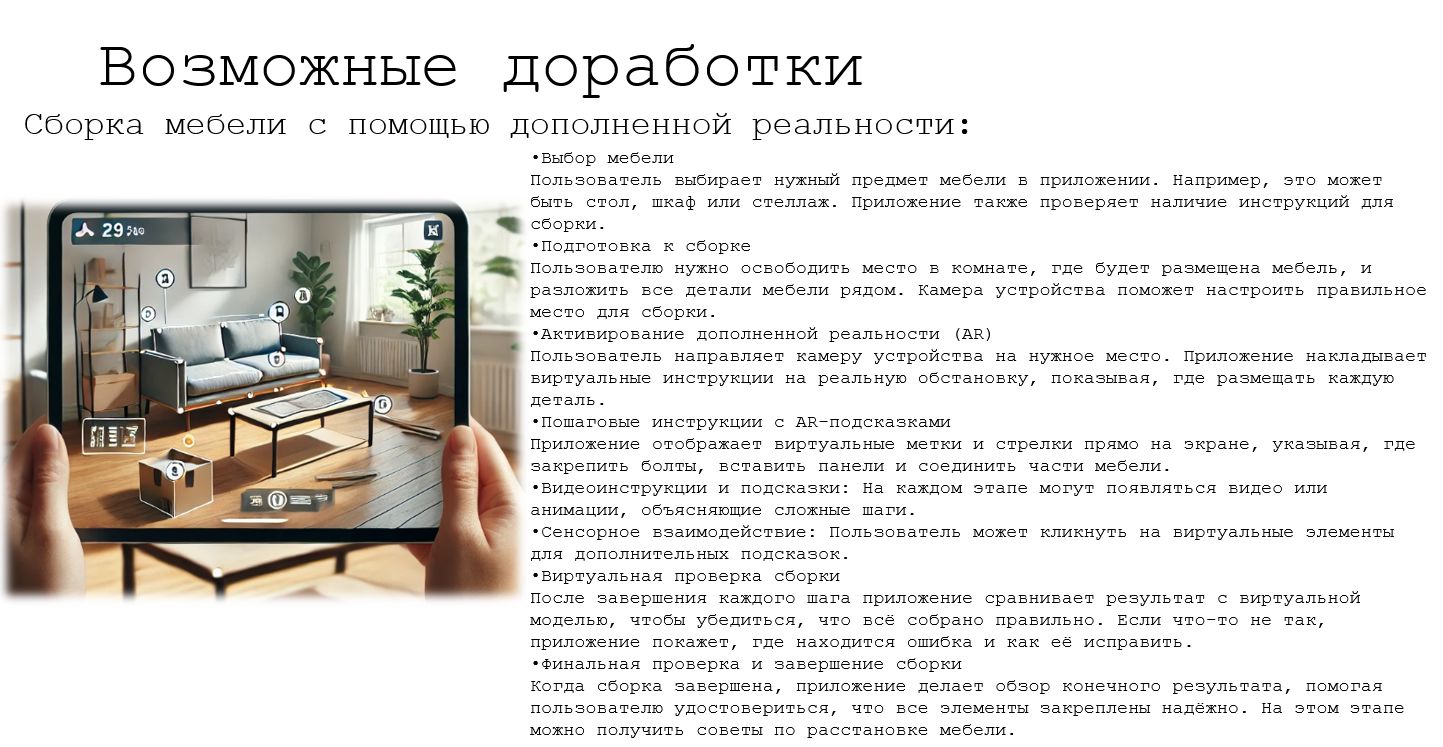


Рисунок 5 – Возможные доработки



Рисунок 6 – Примерный прототип

# 5 Требований к проекту

Программный продукт «Программная система для обустройства интерьера с использованием технологий смешанной и дополненной реальности» предназначен для создания и визуализации интерьера с использованием AR и MR технологий. Система позволяет пользователям настраивать элементы интерьера, проводить виртуальную сборку мебели и создавать пространства с учетом реальных размеров и особенностей.

Программный продукт состоит из следующих подсистем:

1. FE-1 – Учетная запись пользователя

Модуль для регистрации и управления пользовательскими данными.

2. FE-2 – Каталог элементов интерьера

База данных с разнообразными объектами интерьера, такими как мебель, декор и строительные элементы.

3. FE-3 – Визуализация и взаимодействие с объектами в реальном времени

Модуль для отображения выбранных элементов интерьера в реальном времени с использованием технологий дополненной и смешанной реальности.

4. FE-4 – Система рекомендаций по подбору мебели и декора

Алгоритм, который анализирует предпочтения пользователя и предлагает подходящие предметы для дизайна интерьера.

1. FE-5 – Пользовательский интерфейс

Таблица 2. Матрица требований к подсистемам

|  |  |
| --- | --- |
| **Матрица требований** | **Идентификаторы** |
| Операции с учетной записью пользователя | FE-1 UA |
| Просмотр | REQ-UA-1 |
| Операции с данными «параметры интерьера» учетной записи пользователя | FE-1 BI |
| Редактирование | REQ-BI-1 |
| Удаление | REQ-BI-2 |
| Таблица 2. Продолжение | |
| Операции с данными «личные данные» учетной записи пользователя | FE-1 PD |
| Редактирование | REQ-PD-1 |
| Операции с каталогом элементов интерьера | FE-2 C |
| Просмотр | REQ-C-1 |
| Поиск элементов по категории | REQ-C-2 |
| Поиск элементов по стилю | REQ-C-3 |
| Операции со списком сравнения элементов интерьера | FE-3 CL |
| Просмотр | REQ-CL-1 |
| Добавить элемент | REQ-CL-2 |
| Удалить элемент | REQ-CL-3 |
| Операции со значениями параметров схемы интерьера | FE-4 SM |
| Добавление | REQ-SM-1 |
| Редактирование | REQ-SM-2 |
| Удаление | REQ-SM-3 |
| Операции со схемой интерьера | FE-4 SP |
| Просмотр | REQ-SP-1 |
| Сохранение на локальное хранилище | REQ-SP-2 |
| Пользовательский интерфейс | FE-5 UI |
| Операции всех подсистем в веб-интерфейсе | REQ-UI-1 |

Расшифровка идентификаторов

- REQ-UA-1 – Requirement: User Account, 1st Requirement (Требование: Учетная запись пользователя, 1-е требование)

- REQ-BI-1 – Requirement: Building Interior (Parameters), 1st Requirement (Требование: Параметры интерьера, 1-е требование)

- REQ-BI-2 – Requirement: Building Interior (Parameters), 2nd Requirement (Требование: Параметры интерьера, 2-е требование)

- REQ-PD-1 – Requirement: Personal Data, 1st Requirement (Требование: Личные данные пользователя, 1-е требование)

- REQ-C-1 – Requirement: Catalog, 1st Requirement (Требование: Каталог элементов интерьера, 1-е требование)

- REQ-C-2 – Requirement: Catalog, 2nd Requirement (Требование: Каталог элементов интерьера, 2-е требование)

- REQ-C-3 – Requirement: Catalog, 3rd Requirement (Требование: Каталог элементов интерьера, 3-е требование)

- REQ-CL-1 – Requirement: Comparison List, 1st Requirement (Требование: Список сравнения элементов интерьера, 1-е требование)

- REQ-CL-2 – Requirement: Comparison List, 2nd Requirement (Требование: Список сравнения элементов интерьера, 2-е требование)

- REQ-CL-3 – Requirement: Comparison List, 3rd Requirement (Требование: Список сравнения элементов интерьера, 3-е требование)

- REQ-SM-1 – Requirement: Schema Measures, 1st Requirement (Требование: Мерки схемы интерьера, 1-е требование)

- REQ-SM-2 – Requirement: Schema Measures, 2nd Requirement (Требование: Мерки схемы интерьера, 2-е требование)

- REQ-SM-3 – Requirement: Schema Measures, 3rd Requirement (Требование: Мерки схемы интерьера, 3-е требование)

- REQ-SP-1 – Requirement: Schema Pattern, 1st Requirement (Требование: Схема интерьера, 1-е требование)

- REQ-SP-2 – Requirement: Schema Pattern, 2nd Requirement (Требование: Схема интерьера, 2-е требование)

- REQ-UI-1 – Requirement: User Interface, 1st Requirement (Требование: Пользовательский интерфейс, 1-е требование)

# 6 Разработка архитектуры проекта

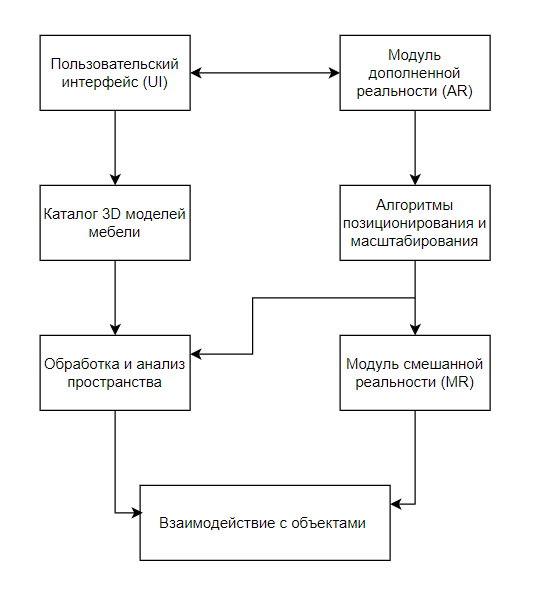
Архитектура программного обеспечения представляет собой основную структуру программной системы, а также процесс создания этих структур и систем. Она включает в себя компоненты программного обеспечения, их взаимодействие, а также характеристики как самих компонентов, так и их связей.   
 Архитектура программной системы аналогична архитектуре здания и служит в качестве плана для системы и процесса разработки, в котором детализированы задачи, которые должны быть выполнены различными группами разработчиков.

**Концептуальная модель**

Концептуальная модель – это совокупность взаимосвязанных понятий, лежащих в основе исследовательского дизайна, системное описание исследуемой области. Концептуальная модель наглядно описывает структуру моделируемой предметной области и связи между ее элементами.

Концептуальная модель помогает решить, какие переменные наиболее важны, какие связи наиболее значимы и, следовательно, какую информацию необходимо собирать.

Схема концептуальной модели показана на рисунке 7.



**Логическая модель**

Целью построения логической модели является получение графического представления логической структуры исследуемой предметной области.

Логическая модель предметной области иллюстрирует сущности, а также их взаимоотношения между собой.

Схема данной модели представлена на рисунке 8.

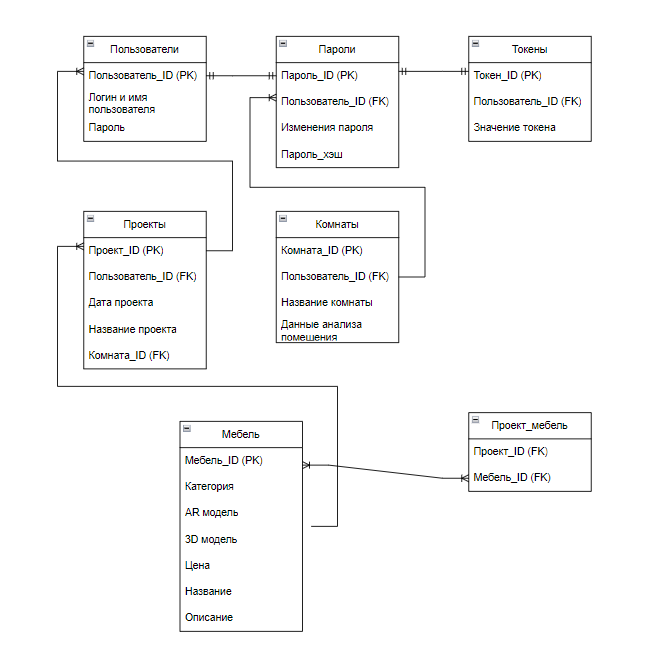


Рисунок 8. ER-диаграмма

На рисунке 9 представлена контекстная диаграмма первого уровня.

**Контекстная диаграмма потоков данных**

Одним из способов представления архитектуры проекта является диаграмма потоков данных.

Диаграммы потоков данных (Data Flow Diagrams – DFD) представляют собой иерархию функциональных процессов, связанных потоками данных.

Цель такого представления – продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

На рисунке 10 представлена диаграмма потоков данных нулевого уровня между подсистемами разрабатываемой системы.



Рисунок 10 - DFD0

На рисунке 11 представлена диаграмма потоков данных первого уровня между подсистемами разрабатываемой системы для модуля обстановки интерьера по параметрам.

Изображение выглядит как снимок экрана, текст, круг, Красочность

Автоматически созданное описание

Рисунок 11 – DFD1

# 7 Измерения проекта

Контроль производственных процессов и результатов является важнейшей задачей современного предприятия, занимающегося разработкой программного обеспечения на заказ. Учитывая особенности такого продукта, как ПО, для оценки эффективности работы и качества итогового продукта применяются специализированные методы.

Набор действий, направленных на измерение и анализ эффективности работы, называется программой измерений.

# Метрики эффективности производственного процесса

* **Problem Resolution Rate (PRR)**

Формула: PRR = Количество дней на обработку задачи

Стратегическая цель: сокращение сроков выполнения проектов по разработке программного обеспечения.

Объект измерения: задачи. Атрибут: время обработки.

Единица измерения: дни.

* **Inspection Fault Density (IFD)**

Формула: IFD = (Количество найденных ошибок / Размер рабочего продукта)

Стратегическая цель: повышение качества создаваемого программного обеспечения.

Объект измерения: процесс инспекции. Атрибут: плотность выявленных ошибок.

Количество ошибок определяется в ходе инспекции.

Единица измерения: ошибки / (страница, требование, строки кода или тест).

# Метрики качества продукта

* **In Process Faults (IPF)**

Формула: IPF = (Количество выявленных ошибок до релиза) / LOC (строки кода)

Стратегическая цель: улучшение качества создаваемого программного обеспечения.

Объект измерения: программный продукт. Атрибут: плотность дефектов.

Количество ошибок определяется на этапе тестирования.

Единица измерения: дефект / строки кода (LOC).

# 9. Разработка рекомендаций по кодированию

Для создания качественного программного обеспечения для обустройства интерьера с использованием технологий коллективной промышленной разработки, следование четким стандартам кодирования крайне важно. Это обеспечит высокую читаемость и поддерживаемость кода, особенно в условиях многокомандной разработки. В данном контексте ключевыми являются стандарты, которые способствуют улучшению процесса разработки в среде Unity с использованием языка C#.

**Unity** — это мощная среда для создания интерактивных приложений и игр, включая системы для проектирования и обустройства интерьера. Важно, чтобы все разработчики следовали единым правилам, чтобы обеспечить хорошую читаемость и легкость в дальнейшей поддержке проекта.

Основные правила кодирования для C# и Unity, которые помогут организовать работу на проекте:

**1. Форматирование**

Отступы и выравнивание: использовать четыре пробела для каждого уровня отступа. В Unity и C# принято использовать отступы для улучшения читаемости, особенно при разработке больших и сложных скриптов, например, для расчёта планировки интерьера.

Пример:

void Start() {

int roomWidth = 5;

int roomLength = 8;

}

Использование фигурных скобок: Каждый блок кода (например, в условиях или циклах) должен начинаться с открывающей фигурной скобки на новой строке.

Пример:

if (roomWidth > 4) {

Debug.Log("Комната достаточно большая");

}

Отступы для объявления методов: Между методами следует оставлять одну пустую строку для лучшей читаемости и разделения логических частей кода.

Пример:

void CreateRoom() {

// код для создания комнаты

}

void SetupFurniture() {

// код для расставления мебели

}

**2. Имена переменных и функций**

Переменные: Имена переменных должны быть информативными, использовать стиль camelCase (первая буква маленькая, последующие — с заглавной), что помогает понять, что переменная представляет. Для переменных, которые содержат несколько слов, разделяйте их с помощью заглавных букв.

Пример:

int roomWidth = 5; // ширина комнаты

string furnitureType = "Sofa"; // тип мебели

Константы: Константы должны быть написаны в верхнем регистре с подчеркиваниями для разделения слов.

Пример:

const float ROOM\_AREA\_THRESHOLD = 20.0f;

Методы и функции: Методы должны быть написаны в PascalCase (каждое слово с заглавной буквы, включая первое).

Пример:

void CreateFurnitureLayout() {

// код для создания планировки мебели

}

**3. Комментарии**

Однострочные комментарии: использовать комментарии для объяснения неочевидных частей кода, особенно если алгоритм расчета имеет сложную логику, например, при размещении мебели или учете пространственных ограничений.

Пример:

// Рассчитываем площадь комнаты для расстановки мебели

float roomArea = roomWidth \* roomLength;

Многострочные комментарии: использовать их для объяснения более сложных блоков кода или для описания класса, его назначения и методов.

Пример:

/\*

Этот класс отвечает за создание и расстановку мебели в комнате,

а также за проверку, подходит ли мебель по размеру.

\*/

class FurniturePlanner {

// Код класса

}

**4. Использование Unity-специфичных практик**

Скрипты и компоненты: В Unity каждое поведение объекта (например, для интерактивных элементов или управления камерой) обычно реализуется в отдельных компонентах-скриптах. Все скрипты следует организовывать логически и использовать стандартные методы, такие как Start(), Awake(), Update().

Пример:

public class RoomManager : MonoBehaviour {

void Start() {

// Инициализация комнаты

}

void Update() {

// Обновление состояния комнаты каждый кадр

}

}

Преимущества использования SerializeField для настройки значений в Unity Editor: это позволяет нам изменять параметры объекта через редактор Unity, не изменяя сам код.

Пример:

[SerializeField] private float roomWidth = 5f; // Ширина комнаты, настраиваемая через Unity Editor

**5. Лучшие практики для командной разработки**

Использование систем контроля версий (Git): В командной разработке крайне важно использовать системы контроля версий, такие как Git, для отслеживания изменений в коде. Следует создавать понятные и краткие коммиты с описанием изменений.

Ревью кода: Регулярные ревью кода помогут команде выявлять потенциальные проблемы, следить за соблюдением стандартов и обмениваться опытом.

Структурирование проекта: структурировать проект в Unity важно с самого начала, разделяя ресурсы и скрипты по папкам, что облегчит навигацию. Например, создать отдельные папки для скриптов, текстур, моделей и сцен.

Пример структуры:

/Assets

/Scripts

/Room

/Furniture

/Models

/Textures

/Scenes

Проверка стиля кода: Для соблюдения единообразного стиля можно использовать автоматические инструменты для проверки C# кода, такие как Resharper или StyleCop. Эти инструменты помогут избежать ошибок в форматировании и несоответствия стандартам.

**6. Производительность**

Для разработки эффективных решений в Unity также важно учитывать производительность, особенно если проект включает сложную 3D-графику и обработку большого количества объектов:

Оптимизация циклов: Постоянно оптимизировать код, например, избегая избыточных вычислений в методах Update(), которые вызываются каждый кадр.

Пул объектов: если необходимо динамически создавать объекты (например, мебель или декор), лучше использовать пулы объектов, чтобы избежать частых операций выделения памяти.