МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра системного анализа и
автоматического управления

# СОВРЕМЕННАЯ АРХИТЕКТУРА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 5 курса 551 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Царёва Даниила Юрьевича

Научный руководитель
ассистент _____ И. А. Люкшин

Заведующий кафедрой
к. ф.-м. н., доцент _____ И. Е. Тананко

Саратов 2023

# ВВЕДЕНИЕ

**Актуальность темы.** В условиях нарастающей динамики современного мира, активного роста инновационных технологий и их всеобщего распространения, а также растущей мобильности общества, современные требования к программному обеспечению значительно возрастают. Воспользоваться качественным продуктом, отвечающем своим функционалом и удобством использования, хотят не только корпорации и компании, но и обычные потребители, желающие быть всегда на связи и в курсе последних событий.

Одним из главных элементов такой доступности является разработка мобильных приложений, способных функционировать на различных платформах и обеспечивать удобную, быструю и качественную работу в самых разных сферах – начиная от бизнеса и развлечений и заканчивая управлением умным домом и финансовыми операциями. В связи с этим, архитектура мобильных приложений крайне важна для полноценного и корректного функционирования программного обеспечения, поскольку современный рынок нуждается в новых разработках, соответствующих высокотехнологическому уровню, решающих глобальные задачи.

Целью настоящей дипломной работы является системное исследование современных архитектурных подходов и технологий в области мобильной разработки, а также их сравнительный анализ с целью выявления преимуществ и возможных недостатков. Для достижения поставленной цели решаются следующие задачи:

– Определить и описать ключевые аспекты современных архитектурных подходов, применяемых в рамках мобильной разработки.

– Проанализировать наиболее распространенные технологии разработки мобильных приложений с точки зрения их применения и возможных ограничений.

– Значимость практической реализации теоретических положений.

Особое внимание будет уделено исследованию современных архитектурных подходов и паттернов проектирования, которые обеспечивают модульность, удобство тестирования и отслеживания ошибок, безопасность и простоту поддержки разрабатываемых приложений. Изучение архитектурных подходов позволит определить динамику прогресса мобильной разработки, выявить ключевые характеристики успешного приложения и разработать

собственную модель архитектуры.

Теоретическая значимость исследования заключается в систематизации основных архитектурных подходов и анализе их возможного влияния на процесс разработки мобильных приложений. Это может стать отправной точкой для дополнительных исследований, а также поддержать принятие решений на уровне проектной работы.

Практическая значимость дипломной работы заключается в расширении знаний в области современной архитектуры мобильной разработки и выборе конкретных технологий и архитектурных подходов. В результате может быть разработано качественное мобильное приложение, удовлетворяющее актуальным требованиям пользователя.

**Цель бакалаврской работы** — изучить и разработать приложения распознавания объектов используя современные архитектурные подходы.

Поставленная цель определила **следующие задачи**:

1. Ознакомиться с основами архитектуры ПО;
2. Изучить паттерны проектирования;
3. Изучить DDD и Clean Architecture архитектуры;
4. Разработать основные модули приложения распознающего объекты;

**Методологические основы** применения архитектурных подходов представлены в работах Р. Мартин [1], Б. Эккель, С. С Исакова, В. Вернон [2], Э. Эванс [3].

**Теоретическая значимость бакалаврской работы**. В ходе выполнения выпускной квалификационной работы было рассмотрены, представлены и реализованы различные архитектуры в рамках мобильного приложения, с помощью которых можно создавать масштабируемые, легковесные мобильные приложения.

**Структура и объем работы**. Бакалаврская работа состоит из введения, 5 разделов, заключения, списка использованных источников и цифрового носителя в качестве приложения. Общий объем работы — 58 страницы, из них 56 страницы — основное содержание, включая 13 рисунков, список использованных источников информации — 20 наименований.

## КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

**Первый раздел «Архитектура как решение проблемы»** посвящен о том что, программное обеспечение может иметь различные проблемы, связанные с разработкой, развертыванием, поддержкой и оптимизацией программных систем. Основные проблемы, которые архитектура помогает решать:

– Большая связанность.
– Плохая масштабируемость.
– Проблема интеграции.
– Сложность изменений.

**Второй раздел «Фундамент архитектуры программного обеспечения»** является ключевым для обеспечения масштабируемости, производительности и стабильности приложения. Учитывая особенности мобильных устройств и их ограничения по мощности, памяти и сетевому подключению, архитектура должна быть оптимизирована и соответствовать сложности задач, которые будет решать приложение.

В подразделе 2.1 рассматривается принцип KISS (Keep It Simple), этот принцип является одним из ключевых принципов проектирования и разработки систем, программного обеспечения и продуктов. Он акцентирует внимание на необходимости создания простых, понятных и легко поддерживаемых решений вместо сложных, запутанных и трудно разбираемых [4].

В подразделе 2.2 рассматривается принцип DRY (Don't Repeat Yourself), этот принцип гласит, что каждая информация или функциональность должна иметь одно и только одно описание, представление или определение в системе. Цель этого принципа – минимизировать повторение кода, избегать дублирования и сократить трудоемкость поддержки и обновления системы.

В подразделе 2.3 рассматривается GRASP (General Responsibility Assignment Software Patterns), это набор рекомендаций и принципов, которые помогают разработчикам в проектировании программных систем, определении объектов и их взаимодействиях. Цель GRASP – создание чистого, модульного и адаптивного кода.

Всего 9 GRASP шаблонов:

1. Создатель (Creator).
2. Информационный эксперт (Information Expert).
3. Контроллер (Controller).

4. Низкая связанность (Low Coupling).

5. Высокая степень зацепления (High Cohesion).

6. Полиморфизм (Polymorphism).

7. Чистая выдумка (Pure Fabrication).

8. Перенаправление (Indirection.

9. Устойчивость к изменениям (Protected Variations).

В подразделе 2.4 рассматриваются принципы SOLID [5] (Single repository, Open/Closed, Liskov, Interface segregation, Dependency Inversion), это акроним, представляющий собой набор из пяти базовых принципов объектно-ориентированного программирования и проектирования, направленных на создание более понятного, гибкого и сопровождаемого кода. SOLID включает следующие принципы:

1. Принцип единственной ответственности (Single Responsibility Principle).

2. Принцип открытости/закрытости (Open/Closed Principle).

3. Принцип подстановки Барбары Лисков (Liskov Substitution Principle).

4. Принцип разделения интерфейсов (Interface Segregation Principle).

5. Принцип инверсии зависимостей (Dependency Inversion Principle).

**Третий раздел «Проблема архитектур мобильных приложений»** посвящен описанию основных проблем приложений, которые возникают при их разработке и поддержке.

В подразделе 3.1 описывается проблема низкой производительности и как архитектура помогает её решить, а именно низкая производительность является одним из ключевых недостатков программного обеспечения, которые имеют значительное влияние на пользователя и его взаимодействие с программой. Важной задачей для разработчиков программного обеспечения является обеспечение определенного уровня производительности и удовлетворение потребностей пользователя, что порой может быть непростой задачей.

В подразделе 3.2 описывается проблема слабой модульности и масштабируемости и как архитектура помогает её решить, слабая модульность означает, что программы не разбиты на отдельные единицы, что требует больших усилий при модификации и обновлении кода. В результате такое программное обеспечение становится труднее поддерживать, теряет гибкость, и его разработка требует более долгого времени. Отметим, что разработчики часто игнорировали принципы модульности, такие концепции, как принцип

единственной ответственности или заменяемости модулей, так как в прошлом программное обеспечение было значительно проще по структуре и функционалу, в отличие от современных приложений.

В подразделе 3.3 описывается проблема сопровождения и разработки и как архитектура помогает её решить, проблемы сопровождения и разработки программного обеспечения являются актуальными и вызывают серьезные затруднения для разработчиков и компаний. Ранее разработка программного обеспечения была трудоемким и длительным процессом, что делало его часто неэффективным и приводило к большому количеству ошибок.

В подразделе 3.4 описывается проблема сложности адаптации и как архитектура помогает её решить, одной из причин этой проблемы является быстрый темп развития технологий, что может сделать трудным для компаний оставаться на переднем крае инноваций. Время и ресурсы, затраченные на изучение и внедрение новых систем, могут быть весьма значительными, что отличает эту проблему от других вызовов бизнеса.

В подразделе 3.5 описывается проблема высокой связности с аппаратной частью и как архитектура помогает её решить, проблема большой связанности с аппаратной частью у мобильных приложений заключается в том, что такие приложения взаимодействуют с аппаратными компонентами устройства (например, сенсоры, GPS, камера, акселерометр, микрофон и т. д.) и тесно связаны с особенностями различных платформ и устройств. Это влияет на архитектуру приложения и его разработку.

**Четвертый раздел «Виды архитектур»** посвящен изучению и реализации различных архитектур приложений, таких как DDD (Domain Driven Design), Clean Architecture, а так же архитектры слоя представления MVP, MVVM, MVI. Архитектура мобильных приложений обычно состоит из различных слоев, схожих с архитектурой веб-приложений. Однако архитектура мобильных приложений также определяется особенностями платформы (Android, iOS). Следует уделить особое внимание выбору наиболее подходящих шаблонов проектирования, решений и инструментов для реализации мобильной архитектуры. Это позволит создать надежное, масштабируемое и поддерживаемое мобильное приложение, адаптированное под нужды пользователей и бизнес-требования.

В подразделе 4.1 DDD (Domain Driven Design) [6] архитектура, её основ-

ные компоненты и её реализацию на уровне мобильного приложения. DDD это подход к разработке программного обеспечения, который ставит акцент на взаимодействие между отдельными частями предметной области (то есть сложности системы) и моделирует их с использованием основных концепций и паттернов.

В подразделе 4.1.1 рассматривается один из основных принципов Domain Driven Design, единый язык. «Единый язык» – это общий словарь терминов и понятий, который разрабатывается командой и используется для описания предметной области (domain) проекта. Единый язык способствует построению совместного понимания между разработчиками, доменными экспертами, тестировщиками и другими участниками команды.

В подразделе 4.1.2 рассматривается один из основных принципов Domain Driven Design, ограниченный контекст. «Ограниченный контекст» (Bounded Context) – это граница, внутри которой определенная модель предметной области (domain) применима и согласована. Ограниченный контекст служит для изоляции аспектов предметной области, гарантируя, что все элементы доменной модели внутри контекста работают согласованно и консистентно.

В подразделе 4.1.3 рассматривается один из основных принципов Domain Driven Design, предметная область. «Предметная область» (Domain) – это область знаний, процессов и функций, которые определены и организованы вокруг цели системы или приложения. Она представляет собой сложную real-world систему, с которой взаимодействует разрабатываемое программное обеспечение, и с которой связаны пользователи и бизнес-правила.

В подразделе 4.1.4 рассматриваются основные паттерны широко применяемы в рамках работы с Domain Driven Design:

1. Паттерн Entity.
2. Паттерн Value Object.
3. Паттерн Repository.
4. Паттерн Factory.
5. Паттерн Domain Event.
6. Паттерн Domain Service.

В подразделе 4.1.5 приводится реализация Domain Driven Design в рамках работы над мобильным приложением для распознавания объектов. Реализация DDD представлена функциональностью профиля пользователя. Ко-

нечный экран отображен на рисунке 1.


Рисунок 1 – Экран профиля

Сама структура DDD в рамках функциональности профиля выглядит следующим образом:

На рисунке 2 представлена структура, верхнеуровнево:

– domain – слой бизнеса, представляет собой сущности и интерфейсы предметной области.

– infrastructure – технический слой, обеспечивает возможность и поддержку реализации остальныхз слоёв. Включает себя обращение к базам данных, схему авторизации и аутентификации и прочее.

– interfaces – слой интерфейсов, представляет собой взаимодействие с внешними системами, такими как, сетевые API интерфейсы, пользовательский интерфейс.

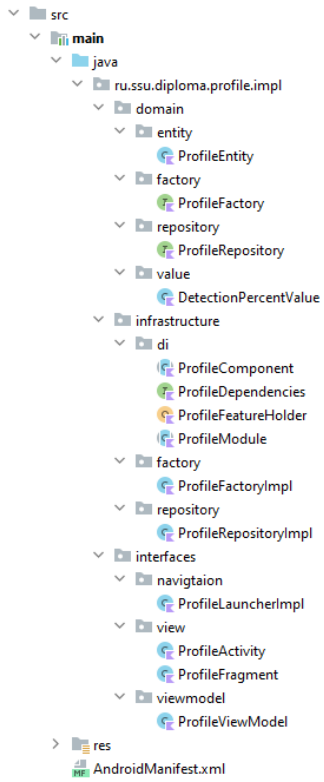


Рисунок 2 – Структура Domain Driven Design

В подразделе 4.1.6 подводится итог рассмотрению, изучению и реализации Domain Driven Design. Domain Driven Design является хорошей архитектурой, позволяющей улучшить качество программного продукта, облегчить его развитие и поддержку, однако полноценная и корректная его реализация в мобильных приложениях часто является избыточной, так как большая часть приложений не обладает сложной логикой, соответственно не все возможности, предоставляемые Domain Driven Design, достигаются в полной мере.

В подразделе 4.2 рассматривается Clean Architecture [7]. Это архитектурный подход, призванный создавать гибкие, масштабируемые и удобные для тестирования системы. Этот термин был популяризирован Робертом Мартином (Uncle Bob) в книге «Чистая архитектура: Руководство по разработке программного обеспечения» (Clean Architecture: A Craftsman's Guide to Software Structure and Design). Основная идея состоит в том, чтобы разделить код на слои, каждый из которых отвечает за выполнение определенной задачи.

В подразделе 4.2.1 рассматривается представление слоёв чистой архитектуры.

– Presentation (Слой представления) — это слой, отвечающий за пред-

ставление данных и взаимодействие с пользователем. Этот слой обеспечивает представление данных в удобном для пользователя виде и обрабатывает пользовательский ввод для обеспечения соответствующих действий в системе.

– Domain (Доменный слой) — это внутренний слой, который содержит основную бизнес-логику приложения, сущности и правила. Целью этого слоя является определение бизнес-правил и ограничений для приложения, а также описание процессов, которые следуют этим правилам.

– Data (Слой данных) — это слой, отвечающий за обработку и управление данными, получаемыми из различных источников, таких как сетевой API-сервис или локальная база данных. Данный слой обеспечивает инкапсуляцию источников данных, что позволяет им легко изменяться без прямого влияния на другие слои приложения.

В подразделе 4.2.2 рассматриваются заблуждения чистой архитектуры. Основные заблуждения это:

– Заблуждения о сущностях (Entities).

– Заблуждения про UseCase/Interactor.

– Заблуждения про маппинг между слоями.

В подразделе 4.2.3 приводится реализация Clean Architecture в рамках работы над мобильным приложением для распознавания объектов. Реализация Clean Architecture представлена функциональностью авторизации пользователя. Конечная структура отображена на рисунке 3. Рассматриваются основные классы ,реализующие полноценное представление чистой архитектуры в мобильном приложении, разложенные по слоям, а именно:

– AuthRepository и его реализация – сущность, выполняющая доступ к API авторизации и инкапсулирующая логику авторизации.

– AuthResult – сущность бизнеса, предоставляющая модель бизнеса авторизации.

– AuthInteractor – сущность инкапсулирующая логику бизнеса.

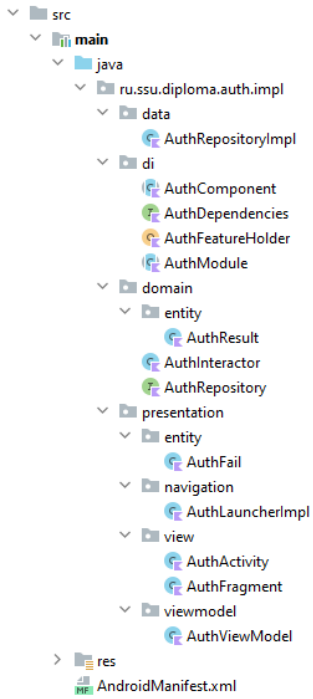


Рисунок 3 – Структура Clean Architecture в модуле авторизации

В подразделе 4.2.4 подводится итог рассмотрению, изучению и реализации Clean Architecture. Clean Architecture является подходом к проектированию мобильных приложений, который предоставляет возможность к созданию модульных, масштабируемых и легко тестируемых приложений, разделяя общую структуру приложения на слои с определенными ответственностями.

В подразделе 4.3 производится сравнение Domain Driven Design и Clean Architecture. В целом, Clean Architecture и Domain Driven Design – это два различных подхода к организации и проектированию системы, которые могут быть использованы вместе или отдельно, в зависимости от конкретных требований и ожиданий от проекта. Однако в контексте мобильных приложений разработчики склоняются больше к Clean Architecture, так как этот проще в реализации и не требует большого ресурса разработки.

В подразделе 4.4 рассматриваются MVx архитектуры, их теоретические основы, назначение и реализация. MVx архитектуры (MVP, MVVM, MVI) решают ряд проблем, связанных с разработкой и поддержкой программного обеспечения. Основные проблемы, которые MVx архитектуры помогают решить, включают:

– Разделение ответственности.

– Модульность.

– Улучшение сопровождаемости и поддержки кода.

– Упрощение совместной разработки.

– Улучшение пользовательского интерфейса.

В подразделе 4.4.1 рассматривается MVP архитектура. MVP (Model-View-Presenter) — это шаблон проектирования, который используется для структурирования пользовательского интерфейса. Целью MVP является отделение бизнес-логики от представления, что облегчает тестирование и повышает модульность приложения. Схема связей представлена на рисунке 4.

Схема связей внутри MVP:

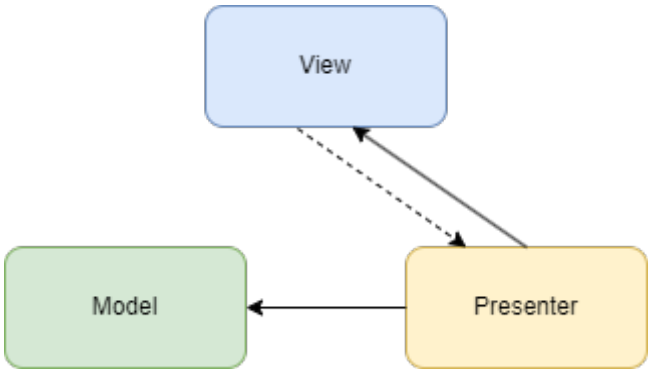


Рисунок 4 – Связи внутри Model-View-Presenter

Основные компоненты:

– Model (Модель) – представляет собой данные и бизнес-логику приложения.

– View (Представление) – отвечает за отображение данных и получение ввода от пользователя.

– Presenter (Презентер) – является связующим звеном между моделью и представлением.

В подразделе 4.4.2 рассматривается MVVM архитектура [8]. MVVM (Model-View-ViewModel) – это шаблон проектирования пользовательского интерфейса, где ViewModel является чётко отделенной от пользовательского представления, а взаимодействие происходит через команды ViewModel. Схема связей представлена на рисунке 5.

Схема связей внутри MVVM:

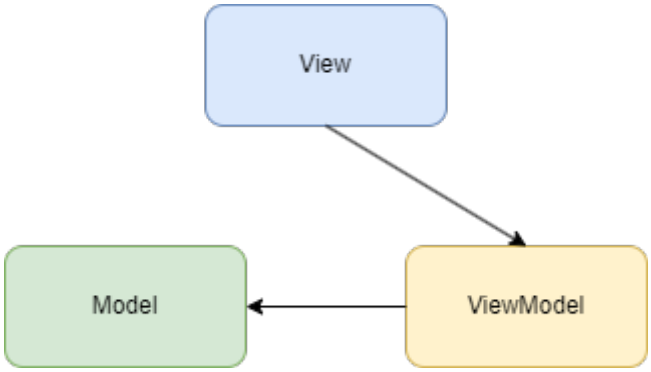


Рисунок 5 – Связи внутри Model-View-ViewModel

Функциональность презентационного слоя в модуле профиля построена на базе архитектурного паттерна MVVM.

В подразделе 4.4.3 рассматривается MVI архитектура [9]. MVI (Model-View-Intent) – это архитектурный паттерн для программирования пользовательского интерфейса, особенно популярный во фреймворке Android. Эта архитектура основана на концепции однонаправленного потока данных [?], что снижает количество неупорядоченных взаимодействий между частями слоя представления и упрощает их тестирование и отладку. Схема связей представлена на рисунке 6.

Схема связей внутри MVI:

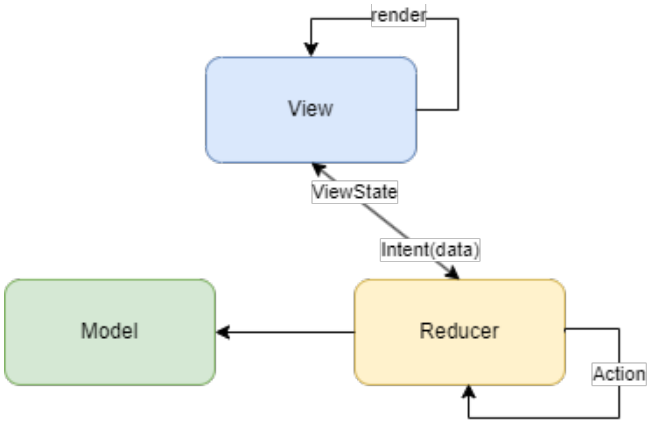


Рисунок 6 – Связи внутри Model-View-Intent

Функциональность презентационного слоя в модуле распознавания объектов построена на базе архитектурного паттерна MVI. В данном случае выбор MVI обусловлен постоянным обновление состояния пользовательского интерфейса. Сам интерфейс представлен на рисунке 7.

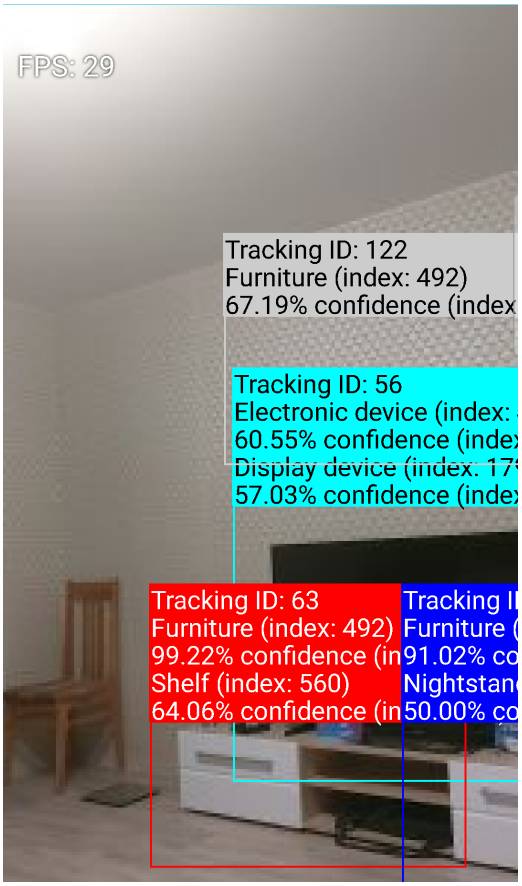


Рисунок 7 – Пользовательский интерфейс функциональности распознавания объектов

В подразделе 4.4.4 происходит сравнение MVx архитектур. Выбор архитектуры слоя представления должен основываться на потребностях в реализации пользовательского опыта и интерфейса. В текущих проектах чаще всего встречаются последние две архитектуры. MVVM более понятен рядовому разработчику и в приоритете на экранах с простым пользовательским интерфейсом. MVI стоит использовать, когда интерфейс сложен и постоянно обновляется.

**Пятый раздел «Преимущества архитектур»** посвящен рассмотрению преимущества архитектур. Архитектуры программных обеспечений предоставляют ряд преимуществ для разработки, поддержки и масштабирования программных продуктов. Некоторые из них включают:

– Модульность.
– Повторное использование.
– Легкость обслуживания.
– Масштабируемость.
– Производительность.

– Безопасность и надежность.

– Гибкость.

## ЗАКЛЮЧЕНИЕ

В данной дипломной работе были изучены и проанализированы следующие вопросы:

– Основные подходы и архитектурные паттерны мобильной разработки, такие как Domain Driven Design, Clean Architecture, архитектуры слоя представления: MVP, MVVM, MVI. Это позволило определить состояние современной архитектуры мобильной разработки и выявить основные преимущества и недостатки каждого рассмотренного подхода.

– Был проведен сравнительный анализ современных подходов к разработке мобильных приложений с использованием различных архитектур. В результате выявлено, что оптимальный подход зависит от специфики проекта, его целей и требований. Таким образом, разработчики и команды должны выбирать наиболее подходящую архитектуру, исходя из уникальных характеристик каждого проекта.

– В качестве практической части работы было разработано мобильное приложение, демонстрирующее успешное применение представленных архитектур. Это позволило подтвердить теоретические положения, сделанные в рамках исследования.

**Основные источники информации:**

1. Босуэлл Д., Фаучин Т., Читаемый код, или Программирование как искусство / Босуэлл Д., Фаучин Т. СПБ: Издательство «Питер», 2012. – 208 с.

2. Холл Г., Адаптивный код. Гибкое кодирование с помощью паттернов проектирования и принципов SOLID / Холл Г. М: Издательство «Диалектика», 2019. – 448 с.

3. Гамма Э., Джонсон Р., Хелм Р., Паттерны объектно-ориентированного проектирования / Гамма Э., Джонсон Р., Хелм Р., СПБ: Издательство «Питер», 2021. – 448 с.

4. Пьер-Ив С., Волшебство Kotlin / Пьер-Ив С. СПБ: Издательство «Питер», 2020. – 536 с.

5. Фримен Э., Бейтс Б., Сьерра К., Фримен Э., Паттерны проектирования / Фримен Э., Бейтс Б., Сьерра К., Фримен Э. СПБ: Издательство «Питер», 2022. – 656 с.

6. Ричардс М., Форд Н., Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы / Ричардс М., Форд Н. СПБ: Издательство «Питер», 2023. – 448 с.

7. Эванс Э. Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем / Эванс Э. М: Издательство «Вильямс», 2018. — 448 с.

8. Вернон В. Реализация методов предметно-ориентированного проектирования / Вернон В. М: Издательство «Диалектика», 2019. – 688 с.

9. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения / Мартин Р. СПБ: Издательство «Питер», 2022. – 352 с.