

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Казанский национальный исследовательский технологический университет»

(ФГБОУ ВО КНИТУ)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**ИУАИТ**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(название института)

Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_АССОИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

по \_\_\_\_\_\_\_\_\_\_учебной\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ практике

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**АССОИ КНИТУ**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(название предприятия, организации, учреждения)

на тему: “Модульное программирование”

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выполнил студент \_\_\_\_\_Као-Ден И.Е.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия И.О., подпись)

Руководитель практики

от предприятия, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

организации, (Фамилия И.О., подпись)

учреждения

Руководитель практики

от кафедры \_\_\_\_\_\_\_Воронина Л.Т.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия И.О., подпись)

Казань \_\_\_\_\_\_\_2025\_\_\_\_\_\_г

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc11541)

[1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ МОДУЛЬНОГО ПРОГРАММИРОВАНИЯ 6](#_Toc15725)

[2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ МОДУЛЬНОГО ПРОГРАММИРОВАНИЯ 9](#_Toc23897)

[3. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ МОДУЛЬНОГО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ GO 12](#_Toc25198)

[ЗАКЛЮЧЕНИЕ 20](#_Toc23954)

[СПИСОК ЛИТЕРАТУРЫ 22](#_Toc25132)

[ПРИЛОЖЕНИЕ 23](#_Toc29373)

# ВВЕДЕНИЕ

Учебная ознакомительная практика представляет собой важный этап в профессиональном становлении будущих специалистов в области информационных технологий и программирования. Это форма учебной деятельности, которая позволяет студентам применить полученные теоретические знания на практике, познакомиться с реальными производственными процессами и приобрести ценные профессиональные навыки. Практика дает возможность погрузиться в рабочую атмосферу, понять специфику будущей профессии и определить направления для дальнейшего профессионального роста.

Основными задачами учебной практики являются закрепление и углубление знаний, полученных в процессе обучения, приобретение практических навыков работы по специальности, изучение организационной структуры предприятия и принципов функционирования его подразделений. Особое значение имеет знакомство с современными технологиями и инструментами, которые используются в профессиональной деятельности. Практика позволяет студенту не только применить свои знания, но и развить такие важные качества, как самостоятельность, ответственность и умение работать в команде.

Практика проходила в период с 30 июня 2025 года по 13 июля 2025 года на базе АССОИ КНИТУ (Ассоциации современных систем обработки информации при Казанском национальном исследовательском технологическом университете). Это учреждение обладает современной материально-технической базой и квалифицированным персоналом, что создает оптимальные условия для прохождения учебной практики. В ходе практики были использованы специализированные компьютерные классы, оснащенные всем необходимым программным обеспечением для разработки и тестирования программных продуктов.

Темой моей учебной практики стало "Модульное программирование". Этот выбор обусловлен особой актуальностью данной методологии в современной разработке программного обеспечения. В условиях постоянно растущей сложности программных систем модульный подход становится не просто удобным, а необходимым инструментом для создания качественного, поддерживаемого и масштабируемого кода. Модульное программирование широко применяется в различных областях - от разработки операционных систем до создания веб-приложений и мобильных приложений.

Основной целью практики было глубокое изучение принципов модульного программирования и их практическое применение. Для достижения этой цели были поставлены следующие задачи: во-первых, необходимо было изучить теоретические основы модульного подхода, включая его преимущества и возможные ограничения; во-вторых, требовалось освоить практические приемы разделения программы на модули в выбранном языке программирования; в-третьих, предстояло разработать небольшое, но функциональное приложение, демонстрирующее все преимущества модульного подхода.

Особое внимание в ходе практики уделялось сравнению модульного и традиционного (монолитного) подходов к программированию. Важно было не только понять технические аспекты реализации модулей, но и осознать, как такой подход влияет на процесс разработки в целом - на скорость написания кода, легкость его тестирования, возможность повторного использования и удобство сопровождения. В качестве практического задания было решено создать универсальный модуль для работы с матричными операциями

Актуальность выбранной темы невозможно переоценить. В современной разработке программного обеспечения практически все крупные проекты создаются с использованием принципов модульности. Это касается как языков программирования общего назначения, так и специализированных систем. Модульный подход позволяет распределять работу между командами разработчиков, упрощает процесс отладки и тестирования, делает код более понятным и легким в сопровождении. Все эти факторы делают изучение модульного программирования крайне важным для будущего специалиста в области информационных технологий.

В ходе прохождения практики были использованы различные методы исследования: изучение специализированной литературы и документации, анализ существующих программных решений, практическое программирование с использованием модульного подхода, тестирование разработанного программного обеспечения. Все это позволило не только получить новые знания, но и сформировать важные практические навыки, которые обязательно пригодятся в будущей профессиональной деятельности.

# 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ МОДУЛЬНОГО ПРОГРАММИРОВАНИЯ

Модульное программирование представляет собой фундаментальный подход к разработке программного обеспечения, который коренным образом изменил принципы создания сложных программных систем. Суть этого подхода заключается в декомпозиции программы на логически завершенные и функционально независимые блоки - модули, каждый из которых выполняет строго определенную задачу. Такой способ организации кода появился как ответ на возрастающую сложность программных продуктов и необходимость управления этой сложностью.

История модульного программирования уходит корнями в 1960-1970-е годы, когда стало очевидно, что традиционные методы разработки монолитных программ перестают удовлетворять требованиям времени. Первые теоретические основы были заложены в работах таких ученых, как Дейкстра и Вирт, а языки программирования Modula и Ada стали первыми реализациями этих идей на практике. Сегодня же принципы модульности пронизывают практически все современные языки программирования и парадигмы разработки.

Ключевой характеристикой модуля является его законченность и слабая связанность с другими частями программы. Хорошо спроектированный модуль можно рассматривать как "черный ящик" - внешнему миру доступен только его интерфейс (набор публичных методов и свойств), в то время как внутренняя реализация скрыта. Это позволяет изменять реализацию модуля, не затрагивая код, который его использует, при условии сохранения интерфейса.

Модульное программирование базируется на нескольких фундаментальных принципах, которые обеспечивают его эффективность. Принцип единственной ответственности требует, чтобы каждый модуль решал только одну четко определенную задачу. Принцип минимальной связанности утверждает, что модули должны взаимодействовать между собой минимально необходимым образом. Принцип сокрытия информации (инкапсуляции) подразумевает, что детали реализации модуля должны быть скрыты от внешнего мира.

Преимущества модульного подхода многочисленны и значительны. Во-первых, он значительно повышает читаемость и понятность кода - вместо одного большого файла с тысячами строк мы имеем набор четко названных модулей, каждый из которых можно анализировать отдельно. Во-вторых, модульная структура упрощает отладку и тестирование, так как каждый модуль можно проверять изолированно от остальной системы. В-третьих, такой подход способствует повторному использованию кода - хорошо написанные модули можно включать в различные проекты без изменений.

Особенно важным преимуществом является то, что модульное программирование позволяет эффективно организовать работу команды разработчиков. Разные специалисты могут параллельно работать над разными модулями, минимально мешая друг другу. Это значительно ускоряет процесс разработки крупных систем и снижает количество конфликтов при слиянии изменений.

Не каждый выделенный блок кода можно считать хорошим модулем. Качественный модуль должен соответствовать ряду важных критериев. Прежде всего, он должен иметь четко определенную и узкую функциональность - если сложно сформулировать, что именно делает модуль в одном предложении, вероятно, он слишком сложен и требует дальнейшего разделения.

Интерфейс модуля (набор предоставляемых им функций и методов) должен быть минимальным, но достаточным для решения его задач. Важно избегать избыточности в интерфейсе, но при этом не делать его слишком ограниченным. Модуль должен быть спроектирован таким образом, чтобы минимизировать зависимости от других модулей - циклические зависимости особенно опасны и должны быть исключены.

Еще одним важным критерием является тестируемость модуля. Хороший модуль можно протестировать изолированно, без необходимости создавать сложную инфраструктуру или запускать всю систему. Это достигается за счет четкого определения зависимостей и возможностью их подмены при тестировании (например, через механизмы dependency injection).

Модульное программирование часто противопоставляется традиционному процедурному (монолитному) подходу, где вся программа представляет собой единое целое. В монолитной программе добавление новых функций часто приводит к необходимости изменять существующий код, что повышает риск появления ошибок. В модульной системе новые функции обычно можно добавить путем создания новых модулей с минимальным воздействием на существующую кодовую базу.

С объектно-ориентированным программированием (ООП) модульное программирование имеет много общего, но это не одно и то же. ООП делает акцент на объектах и их взаимодействии, в то время как модульное программирование фокусируется на организации кода в независимые компоненты. В современных языках эти подходы часто сочетаются - классы группируются в модули (пакеты, пространства имен), что позволяет получить преимущества обоих парадигм.

Функциональное программирование также хорошо сочетается с модульным подходом. Чистые функции (без побочных эффектов) естественным образом ложатся в основу модулей, делая их еще более предсказуемыми и тестируемыми. Многие современные фреймворки (например, для React) успешно используют это сочетание.

# 2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ МОДУЛЬНОГО ПРОГРАММИРОВАНИЯ

Современные языки программирования предлагают разнообразные инструменты для реализации модульного подхода. В Python модульность реализуется через механизм импорта, где каждый файл .py рассматривается как отдельный модуль. Система пакетов позволяет организовывать модули в иерархические структуры. Важной особенностью является специальная переменная \_\_name\_\_, которая помогает различать выполнение модуля как самостоятельной программы или его использование в качестве библиотеки.

В Java модульная система получила серьезное развитие с выходом Java 9, где появилась концепция модулей JPMS (Java Platform Module System). Модуль в Java определяется через специальный файл module-info.java, который явно указывает экспортируемые и требуемые пакеты. Это обеспечивает более строгий контроль зависимостей по сравнению с традиционной системой пакетов.

Язык C использует заголовочные файлы (.h) и компиляцию отдельных единиц трансляции для реализации модульности. Хотя этот механизм более низкоуровневый по сравнению с современными языками, он позволяет достичь хорошей изоляции модулей. В C++ к этому добавляются пространства имен (namespaces), которые помогают избегать конфликтов имен между модулями.

Для эффективной работы с модулями в крупных проектах используются специализированные инструменты управления зависимостями. В экосистеме Python такими инструментами являются pip и Poetry. Они позволяют не только устанавливать внешние зависимости, но и создавать изолированные окружения, где каждый проект может иметь свою версию используемых модулей.

В мире Java основным инструментом стал Maven, который использует концепцию Project Object Model (POM) для описания зависимостей между модулями. Gradle предлагает более гибкую систему на основе DSL. Эти инструменты автоматизируют процесс сборки, разрешают конфликты версий и обеспечивают воспроизводимость сборки.

Особого внимания заслуживают системы контроля версий, такие как Git. Они позволяют организовать хранение модулей в отдельных репозиториях (монорепозиторий vs полирепозиторий) и управлять их версиями через механизмы ветвления и тегирования.

При построении модульных архитектур часто применяются определенные паттерны проектирования. Паттерн Фасад используется для создания упрощенного интерфейса к сложной системе модулей. Паттерн Абстрактная фабрика помогает создавать семейства связанных модулей. Dependency Injection позволяет управлять зависимостями между модулями, делая систему более гибкой и тестируемой.

Особое место занимает паттерн Медиатор, который уменьшает связанность между модулями, перенося логику их взаимодействия в отдельный компонент. Паттерн Наблюдатель часто используется для реализации событийной модели взаимодействия между модулями.

В современных фреймворках популярна концепция Plug-in Architecture, где основное приложение предоставляет точки расширения, а модули реализуют конкретную функциональность. Такой подход широко используется в IDE (Eclipse, Visual Studio Code) и CMS системах (WordPress, Drupal).

Тестирование модульных систем имеет свою специфику. Модульное тестирование (unit testing) фокусируется на проверке отдельных модулей в изоляции. Для этого используются techniques like mocking and stubbing to replace dependencies. Популярные фреймворки включают JUnit для Java, pytest для Python, Google Test для C++.

Интеграционное тестирование проверяет взаимодействие между модулями. Особое внимание уделяется тестированию интерфейсов между модулями и обработке ошибочных ситуаций. Для сложных систем применяется стратегия "сэндвич"-тестирования, сочетающая нисходящий и восходящий подходы.

Контрактное программирование (Design by Contract) формально описывает обязательства модулей перед системой и друг перед другом. Это реализуется через предусловия, постусловия и инварианты, что особенно важно в критически важных системах.

# ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ МОДУЛЬНОГО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ GO

Для демонстрации принципов модульного программирования была поставлена комплексная задача, требующая разработки:

-Универсального модуля для работы с матричными операциями

-Двух различных прикладных программ, использующих этот модуль

Язык Go был выбран не случайно — его система модулей, строгая типизация и ориентация на создание переиспользуемых компонентов идеально подходят для демонстрации преимуществ модульного подхода. Особенностью реализации стало требование создания модуля, который:

-Имеет четко определенную область ответственности

-Предоставляет понятный интерфейс

-Полностью инкапсулирует внутреннюю реализацию

-Может быть протестирован независимо от использующих его программ.

Мы остановились на двух задачах, которые в коде будут реализованы как отдельные программы:

1. Программа \*\*equation\_solver\*\* - решение системы линейных уравнений матричным методом

2. Программа \*\*game\_field\*\* - генерация процедурного игрового ландшафта

Выбор именно этих задач обусловлен их принципиально разной природой при возможности использования общего математического аппарата. Рассмотрим подробнее, почему эти задачи идеально подходят для нашей цели.

Матричные операции, которые лягут в основу общего модуля, являются фундаментальным инструментом в вычислительной математике. Программа equation\_solver будет использовать этот модуль по его прямому назначению - для решения системы уравнений через обращение матрицы коэффициентов. Это классическая задача вычислительной математики, где матричные операции применяются в их "чистом" виде.

В то же время программа game\_field продемонстрирует, как те же самые матричные операции могут служить для совершенно иных целей - в данном случае для представления и обработки игрового пространства. Здесь матрица будет использоваться как двумерная карта высот, где каждое значение соответствует определенному типу местности.

Такой дуализм применения позволит нам показать:

- Как один модуль может обслуживать разные предметные области

- Важность правильного проектирования интерфейсов

- Преимущества инкапсуляции реализации

В кодовой базе это будет организовано следующим образом:

1. Основной модуль \*\*matrixops\*\* (matrixops/matrix.go) будет содержать:

- Тип Matrix для представления двумерных матриц

- Базовые операции: сложение, умножение, транспонирование

- Вспомогательные функции создания и проверки матриц

2. Программа \*\*equation\_solver\*\* (equation\_solver.go) реализует:

- Представление системы уравнений в матричной форме

- Вычисление обратной матрицы

- Решение системы через матричное умножение

- Вывод результатов

3. Программа \*\*game\_field\*\* (game\_field.go) будет включать:

- Генерацию случайной матрицы высот

- Преобразование числовых значений в типы местности

- Визуализацию игрового поля

- Процедурное сглаживание рельефа

Такое разделение позволяет четко продемонстрировать принцип единственной ответственности - каждый компонент системы выполняет строго определенную функцию, а их взаимодействие происходит через четко определенные интерфейсы.

Применение модульного программирования в данном случае даст нам несколько ключевых преимуществ:

Во-первых, мы сможем разрабатывать и тестировать матричные операции независимо от их конечного использования. Модуль matrixops будет содержать автономные тесты, проверяющие корректность реализации всех операций.

Во-вторых, любое улучшение в базовом модуле автоматически скажется на обеих программах. Например, если мы оптимизируем алгоритм матричного умножения, обе программы - и equation\_solver, и game\_field - получат выигрыш в производительности без необходимости внесения изменений в их код.

В-третьих, такая архитектура позволяет легко добавлять новые программы, использующие те же матричные операции. Например, в будущем мы могли бы создать программу для обработки изображений или физического моделирования, повторно используя уже отлаженный модуль matrixops.

Именно эти преимущества делают выбранные задачи идеальными для демонстрации возможностей модульного программирования. В следующих разделах мы подробно рассмотрим реализацию каждого компонента этой системы.

При создании модульных программ на языке Go необходимо строго следовать определенному алгоритму организации кода. Этот процесс включает несколько ключевых этапов, которые обеспечивают корректную работу модулей и их взаимодействие с основной программой.

Организация модульной структуры

Первым шагом в создании модульной программы является правильная организация файловой структуры. Для нашего проекта мы создаем следующую иерархию:

matrix\_project/

├── go.mod

├── matrixops/

│ └── matrix.go

├── equation\_solver.go

└── game\_field.go  
 Корневой каталог проекта содержит файл go.mod, который определяет модуль, подкаталог matrixops с реализацией нашего модуля, и два основных файла программ. Такая структура является стандартной для проектов Go и позволяет компилятору правильно разрешать зависимости.

В корневой директории проекта выполняется команда:

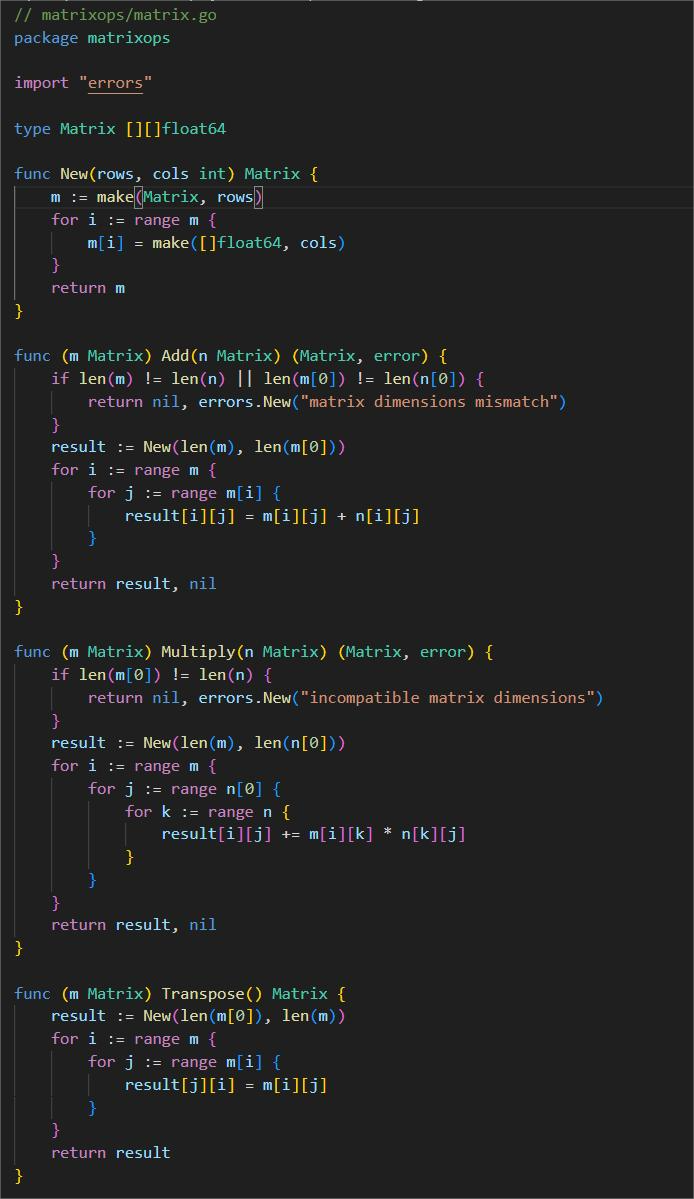
go mod init matrix\_project

Эта команда создает файл go.mod с указанием имени модуля: module matrix\_project

go 1.21

Файл go.mod является обязательным для любого модуля Go и служит для управления зависимостями. При этом важно, чтобы имя модуля (matrix\_project) соответствовало тому, что мы будем использовать в импортах.

matrixops/matrix.go:



Ключевые аспекты реализации:

-Объявление package matrixops в начале файла указывает, что этот код принадлежит модулю matrixops

-Тип Matrix и все методы экспортируются (начинаются с заглавной буквы)

-Каждый метод содержит полную проверку входных параметров

equation\_solver.go:



Особенности подключения модуля:

-Импорт модуля через полный путь: "matrix\_project/matrixops"

-Использование типа matrixops.Matrix для работы с матрицами

-Вызов методов модуля через точку (inv.Multiply(B))

game\_field.go:



Особенности взаимодействия с модулем:

-Использование matrixops.New для создания новых матриц

-Применение методов модуля для основных операций

-Передача и возврат матриц между функциями программы

После реализации всех компонентов, программы запускаются следующими командами:

1. Для решения системы уравнений: go run equation\_solver.go
2. Для генерации игрового поля: go run game\_field.go

Важные аспекты запуска:

-Go автоматически разрешает зависимости благодаря файлу go.mod

-Компилятор находит модуль matrixops по его полному пути

-Все изменения в модуле сразу становятся доступны программам

-Рекомендации по разработке модульных программ

-При создании модульных программ на Go следует учитывать:

-Имена модулей должны быть уникальными и соответствовать пути импорта

-Все экспортируемые элементы модуля должны начинаться с заглавной буквы

-Тесты для модуля следует размещать в файлах с суффиксом \_test.go

-Для обновления зависимостей следует использовать команду go mod tidy

-Версионирование модуля должно следовать семантическому версионированию

-Разработанная структура проекта демонстрирует стандартный подход к организации модульных программ на Go и может служить шаблоном для более сложных проектов. Главное преимущество такого подхода - четкое разделение ответственности между компонентами системы и возможность повторного использования кода.

# ЗАКЛЮЧЕНИЕ

В период с 30.06.2025 по 13.07.2025 была пройдена учебная практика года на базе АССОИ КНИТУ .

В ходе выполнения учебной практики по теме "Модульное программирование" была проделана значительная работа, позволившая глубоко изучить принципы модульного подхода к разработке программного обеспечения. Практическая реализация проекта на языке Go наглядно продемонстрировала преимущества модульной архитектуры и показала, как правильно организовать взаимодействие между отдельными компонентами системы.

Основным результатом работы стало создание универсального модуля для работы с матричными операциями, который успешно был применен в двух различных прикладных задачах. Первая задача — решение системы линейных уравнений — показала, как модульный подход позволяет эффективно решать математические проблемы, изолируя вычислительные алгоритмы в отдельный компонент. Вторая задача — генерация процедурного игрового поля — продемонстрировала гибкость модуля, который смог адаптироваться к совершенно иной предметной области.

Важным достижением стало освоение инструментария языка Go для работы с модулями. В процессе работы были успешно применены такие ключевые команды как go mod init для инициализации модуля, go run для запуска программ, а также отработана практика импортирования модулей через полные пути. Эти навыки имеют фундаментальное значение для профессиональной разработки программного обеспечения на языке Go.

Особую ценность представляет полученный опыт проектирования интерфейсов модуля. Разработанный модуль matrixops демонстрирует все характеристики качественного модуля: четко определенную область ответственности, простой и понятный интерфейс, полную инкапсуляцию внутренней реализации, а также надежную обработку ошибок. Эти принципы могут быть применены при создании любых модулей, независимо от языка программирования.

Практика также позволила оценить преимущества модульного подхода в реальных условиях. Среди наиболее значимых можно отметить: сокращение времени разработки второй программы благодаря повторному использованию кода, упрощение тестирования отдельных компонентов системы, а также возможность параллельной работы над разными частями проекта.

Полученные знания и навыки имеют большую практическую ценность для будущей профессиональной деятельности. Принципы модульного программирования применимы в самых различных областях разработки программного обеспечения — от системного программирования до веб-разработки. Освоенные методики организации кода помогут создавать более структурированные, поддерживаемые и масштабируемые программные продукты.

В заключение следует отметить, что поставленные цели учебной практики были полностью достигнуты. Удалось не только изучить теоретические основы модульного программирования, но и применить эти знания на практике, создав работоспособную систему из взаимодействующих модулей. Этот опыт станет прочной основой для дальнейшего профессионального роста в области разработки программного обеспечения.

# СПИСОК ЛИТЕРАТУРЫ

1. The Go Programming Language Specification – URL: https://go.dev/ref/spec

2. Go Modules Reference – URL: https://go.dev/doc/modules/managing-dependencies

3. Kernighan B., Donovan A. The Go Programming Language. – Addison-Wesley Professional, 2016. – 380 p.

4. Макконнелл С. Совершенный код. – М.: Русская редакция, 2019. – 896 с.

5. Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – Prentice Hall, 2017. – 432 p.

6. Matrix Operations // GeeksforGeeks. – URL: https://www.geeksforgeeks.org/matrix-operations/

7. Valsorda F. Go Module Patterns – URL: https://research.swtch.com/vgo

8. Using Go Modules // The Go Blog. – URL: https://go.dev/blog/using-go-modules

9. About monorepos // GitHub Documentation. – URL: https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories

10. Go by Example: Interfaces – URL: https://gobyexample.com/interfaces

11. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2020. – 366 с.

12. Wirth N. Algorithms + Data Structures = Programs. – Prentice Hall, 1976. – 381 p.

13. IEEE Standard Glossary of Software Engineering Terminology: IEEE Std 610.12-1990. – IEEE, 1990. – 84 p.

# ПРИЛОЖЕНИЕ

|  |  |  |
| --- | --- | --- |
| Код | Описание компетенции | Страницы ПЗ |
| ОПК-1 | Способен применять естественнонаучные и общеинженерные знания, методы математического анализа и моделирования, теоретического и экспериментального исследования в профессиональной деятельности  ОПК-1.1. Знает основы высшей математики, физики, химии, основы вычислительной техники и программирования  ОПК-1.2. Умеет решать стандартные профессиональные задачи с применением естественнонаучных и общеинженерных знаний, методов математического анализа и моделирования  ОПК-1.3. Владеет навыками теоретического и экспериментального исследования объектов профессиональной деятельности | 12 |
| ОПК-7 | Способен участвовать в настройке и наладке программно-аппаратных комплексов  ОПК-7.1. Знает методику настройки и наладки программно-аппаратных комплексов | 14-15 |
| ОПК-8 | Способен разрабатывать алгоритмы и программы, пригодные для практического применения  ОПК-8.1. Знает основные языки программирования и работы с базами данных, операционные системы и оболочки, современные программные среды разработки информационных систем и технологий  ОПК-8.2. Умеет применять языки программирования и работы с базами данных, современные программные среды разработки информационных систем и технологий для автоматизации бизнес-процессов, решения прикладных задач различных классов, ведения баз данных и информационных хранилищ  ОПК-8.3. Владеет навыками программирования, отладки и тестирования прототипов программно-технических комплексов задач | 12 |