Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Крымский федеральный университет имени В. И. Вернадского» (ФГАОУ ВО «КФУ им. В. И. Вернадского»)

Физико–технический институт

Кафедра компьютерной инженерии и моделирования

**РЕФЕРАТ**

по дисциплине «**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ**»

на тему:

«Паттерны проектирования ПО»

Выполнил: обучающийся 2 курса

ПИ-б-о-211

Михайлусов А.М.

Симферополь 2023

# «СОДЕРЖАНИЕ»

Содержание

[«СОДЕРЖАНИЕ» 2](#_Toc138276711)

[«ВВЕДЕНИЕ» 2](#_Toc138276712)

[«ОСНОВНАЯ ЧАСТЬ» 3](#_Toc138276713)

[Что такое паттерн проектирования? 3](#_Toc138276714)

[Основные типы паттернов: 4](#_Toc138276715)

[Отношения между паттернами 5](#_Toc138276716)

[Порождающие паттерны: 6](#_Toc138276717)

[Структурные паттерны: 7](#_Toc138276718)

[Поведенческие паттерны: 8](#_Toc138276719)

[Различные мнения по отношению к паттернам: 10](#_Toc138276720)

[«ЗАКЛЮЧЕНИЕ» 12](#_Toc138276721)

[Выводы: 13](#_Toc138276722)

[Проблемы, которые высветились в ходе работы над рефератом: 14](#_Toc138276723)

[«СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ» 15](#_Toc138276724)

# «ВВЕДЕНИЕ»

Паттерны проектирования являются важным инструментом в разработке программного обеспечения. Они представляют собой повторяемые решения для типичных проблем, возникающих при проектировании и построении сложных систем. Паттерны проектирования помогают разработчикам создавать гибкие, расширяемые и поддерживаемые системы, снижая зависимость от конкретных реализаций и обеспечивая высокую степень переиспользуемости кода [1].

В данном реферате мы рассмотрим различные структуры паттернов проектирования. Первая группа - порождающие паттерны, которые абстрагируют процесс инстанцирования классов и объектов. Они включают в себя абстрактную фабрику, строителя, фабричный метод, прототип и одиночку. Вторая группа - структурные паттерны, которые описывают способы организации классов и объектов в более крупные структуры. Среди них адаптер, мост, компоновщик, декоратор, фасад, приспособленец и заместитель. И, наконец, третья группа - поведенческие паттерны, которые определяют взаимодействие между классами и объектами. Они включают цепочку обязанностей, команду, интерпретатор, итератор, посредника, хранитель, наблюдатель, состояние, стратегию, шаблонный метод и посетитель [1].

В реферате мы также рассмотрим различные точки зрения на использование паттернов проектирования. Мы исследуем практические примеры их применения в реальных проектах, а также обсудим преимущества и недостатки использования паттернов. Мы рассмотрим рекомендации по выбору подходящего паттерна для конкретной задачи и сравним паттерны с другими инструментами разработки.

# «ОСНОВНАЯ ЧАСТЬ»

## Что такое паттерн проектирования?

В общем случае паттерн состоит из четырех основных элементов:

Имя. Указывая имя, мы сразу описываем проблему проектирования, ее решения и их последствия — и все это в одном-двух словах. Присваивание паттернам имен расширяет наш «словарный запас» проектирования и позволяет проектировать на более высоком уровне абстракции. Наличие словаря паттернов позволяет обсуждать их с коллегами, в документации и даже с самим собой. Имена позволяют анализировать дизайн системы, обсуждать его достоинства и недостатки с другими. Нахождение хороших имен было одной из самых трудных задач при составлении каталога [1].

Задача. Описание того, когда следует применять паттерн. Описание объясняет задачу и ее контекст. Может описываться конкретная проблема проектирования, например способ представления алгоритмов в виде объектов. В нем могут быть отмечены структуры классов или объектов, типичные для негибкого дизайна. Также может включаться перечень условий, при выполнении которых имеет смысл применять данный паттерн [1].

Решение. Описание элементов дизайна, отношений между ними, их обязанностей и взаимодействий между ними. В решении не описывается конкретный дизайн или реализация, поскольку паттерн — это шаблон, применимый в самых разных ситуациях. Вместо этого дается абстрактное описание задачи проектирования и ее возможного решения с помощью некоего обобщенного сочетания элементов (в нашем случае классов и объектов) [1].

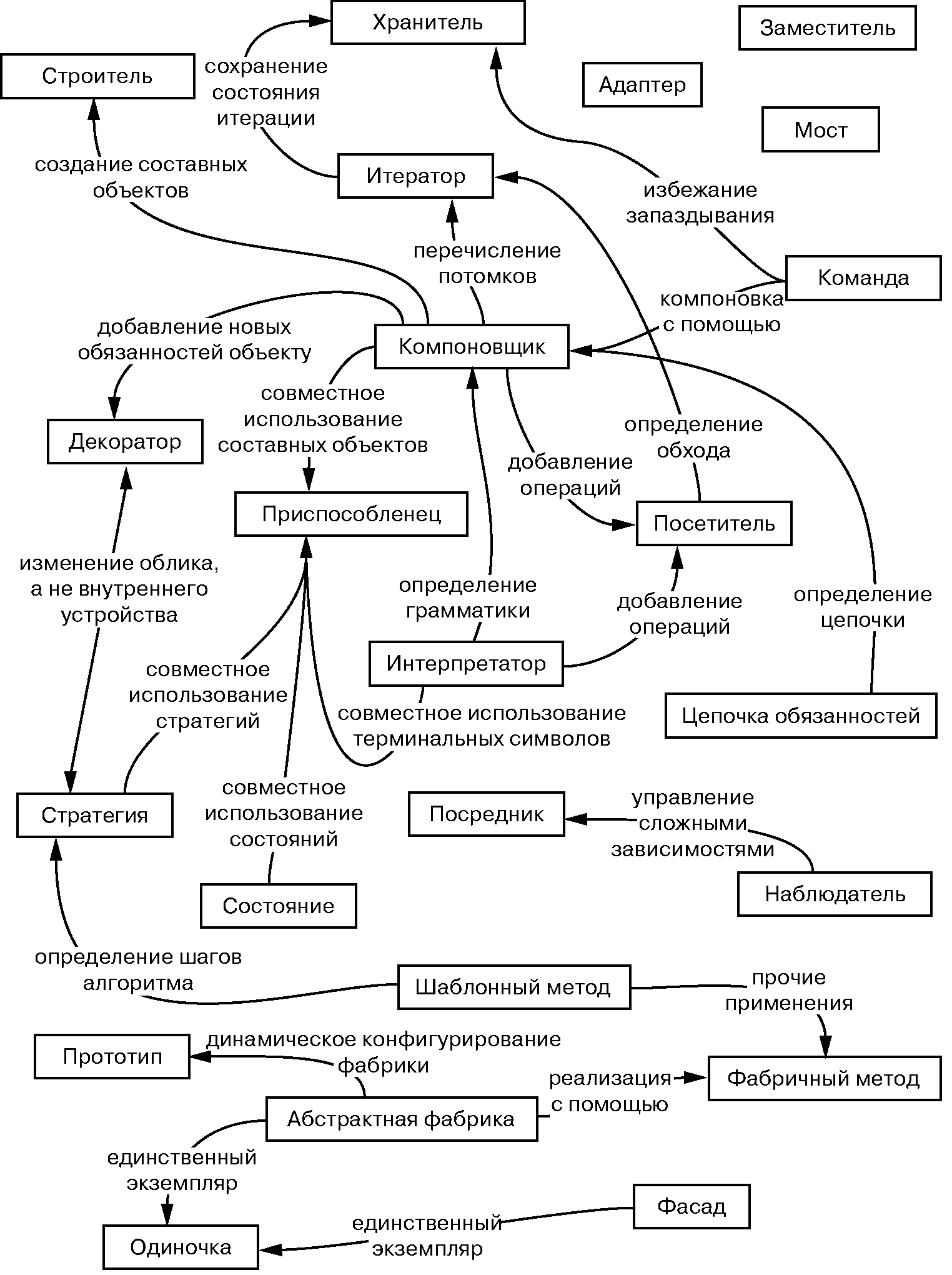
Результаты — следствия применения паттерна, его вероятные плюсы и минусы. Хотя при описании проектных решений о последствиях часто не упоминают, знать о них необходимо, чтобы можно было выбрать между различными вариантами и оценить преимущества и недостатки данного паттерна. Нередко к результатам относится баланс затрат времени и памяти, а также речь может идти о выборе языка и подробностях реализации. Поскольку в объектно-ориентированном проектировании повторное использование зачастую является важным фактором, то к результатам следует относить и влияние на степень гибкости, расширяемости и переносимости системы. Перечисление всех последствий поможет вам понять и оценить их роль [1].

Паттерны проектирования — это не то же самое, что связанные списки или хештаблицы, которые можно реализовать в виде класса и повторно использовать без каких бы то ни было модификаций. С другой стороны, это и не сложные предметно-ориентированные решения для целого приложения или подсистемы [1].

## Основные типы паттернов:

* Порождающие паттерны — паттерны, которые абстрагируют процесс инстанцирования или, иными словами, процесс порождения классов и объектов.
* Структурные паттерны - рассматривают, как классы и объекты образуют более крупные структуры - более сложные по характеру классы и объекты.
* Поведенческие паттерны - определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение.

### Отношения между паттернами



### Порождающие паттерны:

1. Абстрактная фабрика (Abstract Factory) порождающий шаблон проектирования, предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов. Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы (например, для оконного интерфейса он может создавать окна и кнопки). Затем пишутся классы, реализующие этот интерфейс.
2. Строитель (Builder) Строитель(Builder) — порождающий шаблон проектирования предоставляет способ создания составного объекта. Отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления.
3. Фабричный метод (Factory Method) порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.
4. Прототип (Prototype) Задаёт виды создаваемых объектов с помощью экземпляра-прототипа и создаёт новые объекты путём копирования этого прототипа. Он позволяет уйти от реализации и позволяет следовать принципу «программирование через интерфейсы». В качестве возвращающего типа указывается интерфейс/абстрактный класс на вершине иерархии, а классы-наследники могут подставить туда наследника, реализующего этот тип. Проще говоря, это паттерн создания объекта через клонирование другого объекта вместо создания через конструктор.
5. Одиночка (Singleton) порождающий шаблон проектирования, гарантирующий, что в однопоточном приложении будет единственный экземпляр класса с глобальной точкой доступа.

### Структурные паттерны:

1. Адаптер (Adapter) структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс.
2. Мост (Bridge) структурный шаблон проектирования, используемый в проектировании программного обеспечения чтобы «разделять абстракцию и реализацию так, чтобы они могли изменяться независимо». Шаблон мост использует инкапсуляцию, агрегирование и может использовать наследование для того, чтобы разделить ответственность между классами.
3. Компоновщик (Composite) структурный шаблон проектирования, объединяющий объекты в древовидную структуру для представления иерархии от частного к целому. Компоновщик позволяет клиентам обращаться к отдельным объектам и к группам объектов одинаково
4. Декоратор (Decorator) структурный шаблон проектирования, предназначенный для динамического подключения дополнительного поведения к объекту. Шаблон Декоратор предоставляет гибкую альтернативу практике создания подклассов с целью расширения функциональности.
5. Фасад (Facade) структурный шаблон проектирования, позволяющий скрыть сложность системы путём сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы.
6. Приспособленец (Flyweight) структурный шаблон проектирования, при котором объект, представляющий себя как уникальный экземпляр в разных местах программы, по факту не является таковым.
7. Заместитель (Proxy) структурный шаблон проектирования, который предоставляет объект, который контролирует доступ к другому объекту, перехватывая все вызовы (выполняет функцию контейнера).

### **Поведенческие паттерны:**

1. Цепочка обязанностей (Chain of responsibility) (Chain of responsibility) — поведенческий шаблон проектирования, предназначенный для организации в системе уровней ответственности.
2. Команда (Command) поведенческий шаблон проектирования, используемый при объектно-ориентированном программировании, представляющий действие. Объект команды заключает в себе само действие и его параметры.
3. Интерпретатор (Interpreter) поведенческий шаблон проектирования, решающий часто встречающуюся, но подверженную изменениям, задачу. Также известен как Little (Small) Language
4. Итератор (Iterator) интерфейс, предоставляющий доступ к элементам коллекции (массива или контейнера) и навигацию по ним. В различных системах итераторы могут иметь разные общепринятые названия. В терминах систем управления базами данных итераторы называются курсорами.
5. Посредник (Mediator) поведенческий шаблон проектирования, обеспечивающий взаимодействие множества объектов, формируя при этом слабую связанность и избавляя объекты от необходимости явно ссылаться друг на друга
6. Хранитель (Memento) поведенческий шаблон проектирования, позволяющий, не нарушая инкапсуляцию, зафиксировать и сохранить внутреннее состояние объекта так, чтобы позднее восстановить его в это состояние.
7. Наблюдатель (Observer) поведенческий шаблон проектирования. Также известен как «подчинённые» (Dependents). Создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними.
8. Состояние (State) поведенческий шаблон проектирования. Используется в тех случаях, когда во время выполнения программы объект должен менять своё поведение в зависимости от своего состояния.
9. Стратегия (Strategy) поведенческий шаблон проектирования, предназначенный для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путём определения соответствующего класса. Шаблон Strategy позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют.
10. Шаблонный метод (Template method) поведенческий шаблон проектирования, определяющий основу алгоритма и позволяющий наследникам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом.
11. Посетитель (Visitor) поведенческий шаблон проектирования, описывающий операцию, которая выполняется над объектами других классов. При изменении visitor нет необходимости изменять обслуживаемые классы.

### Различные мнения относительно паттернов:

Паттерны появились потому, что многие разработчики искали пути повышения гибкости и степени повторного использования своих программ.

Использование паттернов при разработке программных систем позволяет проектировщику перейти на более высокий уровень разработки проекта. Теперь архитектор и программист могут оперировать образными названиями паттернов и общаться на одном языке.

Прежде всего, опытный разработчик понимает, что не нужно решать каждую новую задачу с нуля. Вместо этого он старается повторно воспользоваться теми решениями, которые оказались удачными в прошлом. Отыскав хорошее решение один раз, он будет прибегать к нему снова и снова. Именно благодаря накопленному опыту проектировщик и становится экспертом в своей области. Во многих объектно-ориентированных системах встречаются повторяющиеся паттерны, состоящие из классов и взаимодействующих объектов. С их помощью решаются конкретные задачи проектирования, в результате чего объектно-ориентированная архитектура становится более гибкой, элегантной, и может использоваться повторно. Проектировщик, знакомый с паттернами, может сразу же применять их к решению новой задачи, не пытаясь каждый раз изобретать велосипед.

Если вы знаете паттерн, многие проектировочные решения далее следуют автоматически.

Паттерны проектирования упрощают повторное использование удачных проектных и архитектурных решений. Представление прошедших проверку временем методик в виде паттернов проектирования делает их более доступными для разработчиков новых систем. Паттерны проектирования помогают выбрать альтернативные решения, упрощающие повторное использование системы, и избежать тех альтернатив, которые его затрудняют. Паттерны улучшают качество документации и сопровождения существующих систем, поскольку они позволяют явно описать взаимодействия классов и объектов, а также причины, по которым система была построена так, а не иначе. Проще говоря, паттерны проектирования дают разработчику возможность быстрее найти правильный путь.

# «ЗАКЛЮЧЕНИЕ»

В ходе нашего реферата мы рассмотрели различные категории паттернов проектирования, включая порождающие, структурные и поведенческие паттерны. Мы изучили их основные принципы и примеры применения, а также обсудили их практическую значимость.

Однако, необходимо помнить, что паттерны проектирования не являются универсальным решением для всех задач. Их применение требует внимательного анализа конкретной ситуации и оценки выгоды, которую они могут принести. Неконтролируемое использование паттернов может привести к излишней сложности и избыточности кода.

Важно также отметить, что паттерны проектирования не являются единственным инструментом разработки программного обеспечения. Они должны рассматриваться в контексте других принципов и подходов, таких как SOLID-принципы, тестирование и рефакторинг кода.

В заключении, паттерны проектирования являются мощным инструментом, который помогает разработчикам создавать эффективное и поддерживаемое программное обеспечение. Их изучение и применение требует время и практики, но может значительно улучшить качество и структуру разрабатываемых систем.

# Выводы:

1. Паттерны проектирования представляют собой повторяемые решения для типичных проблем, возникающих при проектировании программного обеспечения.
2. Паттерны проектирования помогают создавать гибкие, расширяемые и поддерживаемые системы, снижая зависимость от конкретных реализаций и обеспечивая высокую степень переиспользуемости кода.
3. Порождающие паттерны абстрагируют процесс инстанцирования классов и объектов, включая абстрактную фабрику, строителя, фабричный метод, прототип и одиночку.
4. Структурные паттерны определяют способы организации классов и объектов в более крупные структуры, включая адаптер, мост, компоновщик, декоратор, фасад, приспособленец и заместитель.
5. Поведенческие паттерны определяют взаимодействие между классами и объектами, включая цепочку обязанностей, команду, интерпретатор, итератор, посредника, хранитель, наблюдатель, состояние, стратегию, шаблонный метод и посетитель.
6. Паттерны проектирования не являются универсальным решением для всех задач. Их применение требует анализа конкретной ситуации и оценки выгоды, которую они могут принести.
7. Неконтролируемое использование паттернов может привести к излишней сложности и избыточности кода.
8. Паттерны проектирования должны рассматриваться в контексте других принципов и подходов, таких как SOLID-принципы, тестирование и рефакторинг кода.
9. Изучение и применение паттернов проектирования требует времени и практики, но может значительно улучшить качество и структуру разрабатываемых систем.

# Проблемы, которые высветились в ходе работы над рефератом:

1. Проблема переизбытка абстракции при работе со множеством паттерном и как это сказывается на понимании программистом написанного кода.
2. Проблема возможного недостатка производительности при использовании динамического связывания, необходимого для реализации большинства паттернов.
3. Проблема повсеместного использования паттернов.

# «СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ»

1. "Паттерны проектирования: элементы повторного использования объектно-ориентированного программирования". Авторы: Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес. Издательство: Addison-Wesley Professional. С. 395 ISBN: 978-0201633610
2. "Паттерны проектирования. Построение гибких, надежных и расширяемых приложений". Авторы: Эрик Фриман, Элизабет Робсон, Берт Бейтс, Кэти Сьерра. Издательство: O'Reilly Media. С. 694 ISBN: 978-1491950296.
3. "Паттерны проектирования: практическое руководство". Авторы: Алан Шэллоуэй, Джеймс Р. Тротт. Издательство: Pearson Education. Страницы: 240. ISBN: 978-0321127426.
4. "Паттерны архитектуры предприятия". Автор: Мартин Фаулер. Издательство: Addison-Wesley Professional. С. 560. ISBN: 978-0321127426.
5. "Проектирование с использованием паттернов DDD: управление сложностью в центре программного обеспечения". Автор: Эрик Эванс. Издательство: Addison-Wesley Professional. С. 560. ISBN: 978-0321127426.
6. "Рефакторинг: улучшение проекта существующего кода". Автор: Мартин Фаулер. Издательство: Addison-Wesley Professional. С. 431 ISBN: 978-0321127426.