Свищёв А. В., Кравцова Е. Ю. Анализ архитектурных шаблонов проектирования для конструирования программного обеспечения // Международный журнал информационных технологий и энергоэффективности. – 2023. – Т. 8 № 4(30) с. 49–54

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.415.2

АНАЛИЗ АРХИТЕКТУРНЫХ ШАБЛОНОВ ПРОЕКТИРОВАНИЯ ДЛЯ КОНСТРУИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

[1]Свищёв А. В., [2]Кравцова Е. Ю.
МИРЭА - Российский технологический университет, Москва, Россия (119454, г. Москва, пр. Вернадского, 78), e-mail: [1]svicshyov@mirea.ru, [2]9067320378@mail.ru

Данная статья дает представление о том, как выбираются архитектурные шаблоны проектирования с точки зрения конструирования программного обеспечения. С учетом всех существующих архитектурных шаблонов может быть трудно решить, какой именно выбрать. Даже если архитектурный шаблон описывает, в каких обстоятельствах его следует применять, это не обязательно приводит к тому, что он применяется, когда появляется такая возможность. Это может быть связано с факторами, которые гораздо более очевидны в процессе разработки, такими как: время, капитал и сложность реализации.

Ключевые слова: Архитектурные шаблоны проектирования, конструирования программного обеспечения, проектирование архитектуры.

ANALYSIS OF ARCHITECTURAL DESIGN PATTERNS FOR SOFTWARE DESIGN

[1] Svishchev A. V., [2] Kravtsova E. Y.
MIREA - Russian Technological University, Moscow, Russia (119454, Moscow, Vernadskogo Ave., 78), e-mail: [1]svicshyov@mirea.ru, [2]9067320378@mail.ru

This article provides insight into how architectural design patterns are chosen from a software design perspective. With all the existing architectural patterns available, it can be difficult to decide which one to choose. Even if an architectural template describes in what circumstances it should be applied, this does not necessarily result in it being applied when the opportunity arises. This may be due to factors that are much more obvious in the development process, such as: time, capital, and implementation complexity.

Keywords: Architectural design patterns, software design, architecture design.

В последние десятилетия программное обеспечение стало повсеместным. Почти все современные инженерные системы включают в себя важные программные подсистемы; это включает системы в транспортном, финансовом, образовательном, здравоохранительном, юридическом, военном и деловом секторах. Наряду с увеличением полезности программного обеспечения, его возможностей, стоимости и размера наблюдается соответствующий рост методов, моделей, инструментов, показателей и стандартов, поддерживающих разработку программного обеспечения.

Конструирование программного обеспечения — это процесс преобразования пользовательских требований в некоторую подходящую форму, которая помогает разработчику в написании кода и реализации программного обеспечения.

Для оценки требований пользователя создается документ SRS (Спецификация требований к программному обеспечению), тогда как для кодирования и реализации необходимы более конкретные и подробные требования с точки зрения программного обеспечения. Результат этого процесса можно напрямую использовать в реализации на языках программирования.

Конструирование программного обеспечения с использованием архитектурных шаблонов проектирования происходит в несколько этапов. Архитектурные шаблоны представляют собой обобщенные решения для повторяющихся проблем, возникающих при проектировании архитектуры программного обеспечения. Эти шаблоны помогают разработчикам структурировать и организовать код, обеспечивая масштабируемость, гибкость и поддерживаемость. Вот основные этапы конструирования программного обеспечения с использованием архитектурных шаблонов:

1. Анализ требований: сначала необходимо проанализировать требования к программному обеспечению и определить ключевые функциональные и нефункциональные характеристики. Это поможет определить, какие архитектурные шаблоны могут быть наиболее подходящими для проекта.

2. Выбор архитектурных шаблонов: на основе анализа требований выберите подходящие архитектурные шаблоны для вашего проекта. Некоторые распространенные архитектурные шаблоны включают Model-View-Controller (MVC), Model-View-ViewModel (MVVM). Выбор шаблона зависит от типа приложения, технологий и ожидаемой нагрузки.

3. Проектирование архитектуры: С использованием выбранных архитектурных шаблонов разработайте архитектуру вашего программного обеспечения. Определите основные компоненты, их взаимосвязи и обязанности. Убедитесь, что архитектура соответствует требованиям проекта и обеспечивает гибкость для изменений и масштабирования.

4. Реализация компонентов: на этапе реализации начните создавать компоненты программного обеспечения в соответствии с выбранными архитектурными шаблонами. Следуйте принципам разделения обязанностей, инкапсуляции, модульности и повторного использования кода.

В разработке программного обеспечения выделяют три уровня проектирования результатов:

Архитектурный дизайн (Architectural Design). Архитектурный дизайн является высшей абстрактной версией системы. Он идентифицирует программное обеспечение как систему со многими компонентами, взаимодействующими друг с другом. На этом уровне дизайнеры получают представление о предлагаемой области решения.

Высокоуровневый дизайн (High-level Design). Высокоуровневый дизайн разбивает концепцию архитектурного дизайна «единая сущность-множество компонентов» на менее абстрактное представление подсистем и модулей и изображает их взаимодействие друг с другом. Высокоуровневое проектирование фокусируется на том, как система вместе со всеми

ее компонентами может быть реализована в виде модулей. Он распознает модульную структуру каждой подсистемы, а также их взаимосвязь и взаимодействие друг с другом.

Детальный дизайн (Detailed Design). Детальный дизайн касается части реализации того, что рассматривается как система и ее подсистемы в предыдущих двух проектах. Это более подробно относится к модулям и их реализациям. Он определяет логическую структуру каждого модуля и их интерфейсы для связи с другими модулями.

Разобрав уровни проектирования (конструирования) программного обеспечения следующим этапом, производится анализ существующих архитектурных шаблонов проектирования:

Архитектурный шаблон — это высокоуровневый проект того, как элементы или компоненты взаимосвязаны. Он может показывать предопределенные подсистемы, их обязанности и их отношения между собой. Архитектурный шаблон является способом абстрагирования элементов и компонентов, чтобы систему было легче понять, смоделировать и описать. Не нужно показывать, как спроектирован каждый элемент или компонент, только то, как он может быть использован и для какой цели. Это похоже на класс в объектно-ориентированном программировании с публичным интерфейсом, в то время как частный интерфейс не показывается для внешнего мира.

MVC (Model-View-Controller) — является самым ранним архитектурным шаблоном, состоящий из трёх компонентов:

Model — известен как самый низкий уровень, что является базовой бизнес-логикой и данными;

View — компоненты интерфейса для отображения данных. Использует шаблон Observer для обновления модели и отображения обновленной модели при необходимости;

Controller — это компонент обеспечивающий взаимосвязь между оболочкой (View) и моделью (Model). Сюда сначала направляется ввод, который обрабатывает запрос через модель и передает его обратно в представление.

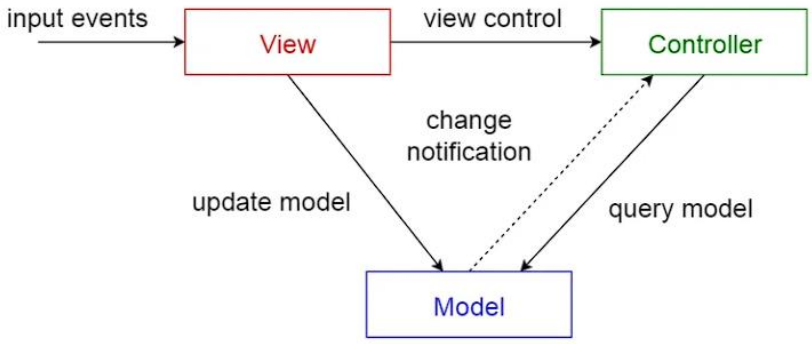

Рисунок 1 – MVC шаблон проектирования

Шаблон проектирования MVC позволяет разделить внешний и внутренний код на отдельные части, чтобы упростить обновление и масштабирование приложения без вмешательства или прерывания. Модель MVC также позволяет нескольким разработчикам одновременно работать над разными частями приложения. Однако существуют риски: представление модели для просмотра может вызвать проблемы с безопасностью и

производительностью. MVC распространен для веб-приложений, библиотек и пользовательских интерфейсов.

Pipe-filter pattern — шаблон, использующийся для структурирования систем, которые производят и обрабатывают поток данных. Каждый шаг обработки заключен в компонент фильтра . Данные для обработки передаются по каналам. Эти каналы можно использовать для буферизации или синхронизации.

Внедряют в рабочие процессы в биоинформатике. Последовательные фильтры выполняют лексический анализ, синтаксический анализ, семантический анализ и генерацию кода.

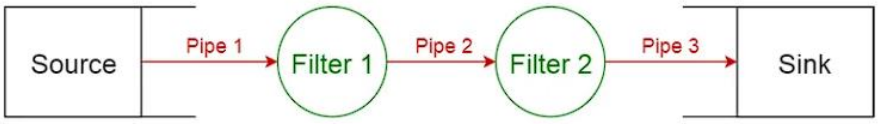

Рисунок 2 – Pipe-filter pattern шаблон проектирования

Данный шаблон позволяет разложить задачу, выполняющую сложную обработку, на ряд отдельных элементов, которые можно использовать повторно. Это может повысить производительность, масштабируемость и возможность повторного использования, позволяя независимо развертывать и масштабировать элементы задачи, выполняющие обработку.

Layered pattern (многоуровневая архитектура) — этот шаблон можно использовать для структурирования программ, которые можно разбить на группы подзадач, каждая из которых находится на определенном уровне абстракции. Каждый уровень предоставляет услуги следующему более высокому уровню.

- Уровень представления (также известный как уровень пользовательского интерфейса);
- Прикладной уровень (также известный как сервисный уровень);
- Уровень бизнес-логики (также известный как уровень домена);
- Уровень доступа к данным (также известный как уровень сохраняемости).

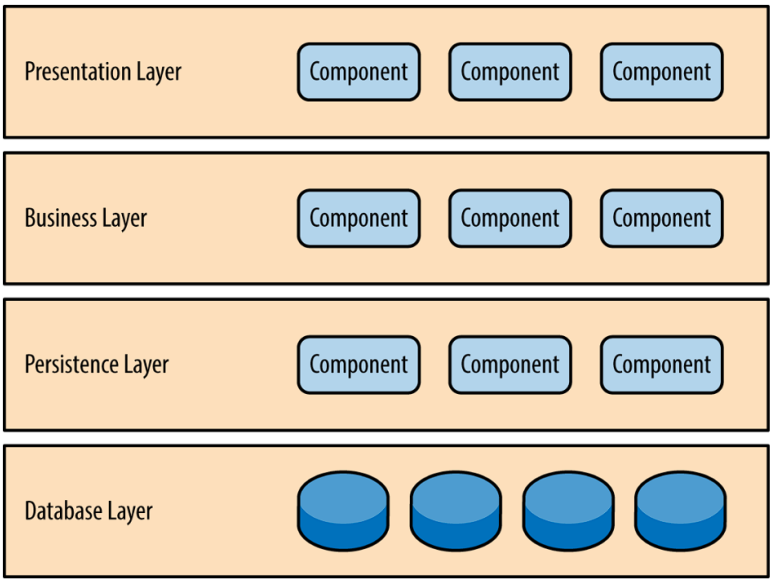

Рисунок 3 – Шаблон многоуровневой архитектуры

Одной из мощных особенностей шаблона многоуровневой архитектуры является разделение задач между компонентами. Компоненты внутри определенного уровня имеют дело только с логикой, относящейся к этому уровню. Например, компоненты на уровне представления имеют дело только с логикой представления, тогда как компоненты, находящиеся на бизнес-уровне, имеют дело только с бизнес-логикой. Этот тип классификации компонентов упрощает создание эффективных моделей ролей и ответственности в вашей архитектуре, а также упрощает разработку, тестирование, управление и обслуживание приложений с использованием этого архитектурного шаблона благодаря четко определенным интерфейсам компонентов и ограниченному объему компонентов.

Таблица 1 – Сравнение анализируемых архитектурных шаблонов

Шаблон	**Преимущества**	**Недостатки**
Layered pattern	Нижние компоненты могут использоваться более высокими. Внедрение компонентов упрощает стандартизацию, поскольку четко определяется уровень компонента. Изменения вносятся внутри компонента, не затрагивая другие	Не может быть универсальным решением
Pipe-filter pattern	Демонстрация параллельной обработки. Легко добавляемые фильтры. Возможность повторно использовать фильтры	Эффективность ограничивается самым медленным процессом фильтрации
MVC	Позволяет быстро создавать несколько представлений одной и той же модели. Отключение и подключение во время выполнения процессов	Может привести ко множеству ненужных обновлений для действий пользователя

Архитектурные шаблоны проектирования играют важную роль в разработке программного обеспечения, так как они предоставляют проверенные временем и обобщенные решения для ряда распространенных проблем, возникающих при проектировании архитектуры. Основываясь на опыте предыдущих проектов, архитектурные шаблоны помогают разработчикам создавать масштабируемые, гибкие и поддерживаемые системы.

Выводы, сделанные в результате исследования архитектурных шаблонов проектирования:

1. Обобщенные решения: Архитектурные шаблоны представляют собой обобщенные решения для типовых проблем, возникающих при проектировании архитектуры программного обеспечения.

2. Структура и организация: Шаблоны помогают структурировать и организовать код, что в свою очередь облегчает разработку, тестирование и поддержку системы.

3. Модульность и разделение обязанностей: Шаблоны обеспечивают разделение обязанностей между различными компонентами системы, что улучшает модульность и упрощает внесение изменений.

4. Повторное использование и стандартизация: Архитектурные шаблоны способствуют повторному использованию кода и стандартизации подходов к разработке, что может сократить время и усилия, затрачиваемые на проект.

5. Улучшение качества и снижение рисков: Применение проверенных архитектурных шаблонов может улучшить качество программного обеспечения и снизить риски, связанные с архитектурными решениями.

Однако важно помнить, что архитектурные шаблоны не являются универсальными решениями для всех ситуаций. При выборе и применении шаблонов необходимо учитывать контекст и требования конкретного проекта, а также возможные ограничения и компромиссы, связанные с использованием того или иного шаблона. Команда разработчиков должна убедиться, что, проведя анализ требований к разрабатываемому продукту полностью вписываются в выбранную архитектуру.

Список литературы

1. Байдыбеков А.А., Гильванов Р.Г., Молодкин И.А. Современные фреймворки для разработки WEB-приложений // Интеллектуальные технологии на транспорте. 2020. URL: https://cyberleninka.ru/article/n/sovremennye-freymvorki-dlya-razrabotki-web-prilozheniy (дата обращения: 28.03.2023).
2. Ихтиар В.Ф. Сравнение кросс-платформенных фреймворков // Вестник магистратуры. 2018. URL: https://cyberleninka.ru/article/n/sravnenie-kross-platformennyh-freymvorkov (дата обращения: 30.03.2023).
3. Бастрикина В.В. Сравнительный анализ адаптивных css фреймворков // Актуальные проблемы авиации и космонавтики. 2016. URL: https://cyberleninka.ru/article/n/sravnitelnyy-analiz-adaptivnyh-css-freymvorkov (дата обращения: 31.03.2023).

References

1. Baidybekov A.A., Gilvanov R.G., Molodkin I.A. Modern frameworks for WEB-applications development // Intelligent Technologies in Transport. 2020. URL: https://cyberleninka.ru/article/n/sovremennye-freymvorki-dlya-razrabotki-web-prilozheniy (accessed 28.03.2023).
2. Ikhtiar V.F. Comparison of cross-platform frameworks // Bulletin of Magistracy. 2018. URL: https://cyberleninka.ru/article/n/sravnenie-kross-platformennyh-freymvorkov (accessed: 30.03.2023).
3. Bastrykina V.V. Comparative analysis of adaptive css frameworks // Actual problems of aviation and cosmonautics. 2016. URL: https://cyberleninka.ru/article/n/sravnitelnyy-analiz-adaptivnyh-css-freymvorkov (accessed: 03/31/2023).