EFFICIENCY OF DESIGN PATTERNS IN SOFTWARE ENGINEERING

ЭФФЕКТИВНОСТЬ ШАБЛОНОВ ПРОЕКТИРОВАНИЯ В ПРОГРАММНОЙ ИНЖЕНЕРИИ

STATNIK Aurel, student, Specialitatea: TI
Academia de Studii Economice din Moldova
Republica Moldova, Chișinău, str. Bănulescu-Bodoni 61, www.ase.md
e-mail autor: aurel.statnyk@gmail.com

Abstract. *Design patterns are always a useful concept that can be used when designing and developing a software application. Performance plays an important role in the quality attribute of an enterprise application. The purpose of this work is to show how important it is to know and be able to apply design patterns in real projects. How can these design patterns affect product performance and quality. This work will also show how the lack of design patterns and other aspects of programming affects the quality of employees' work and their productivity. This study also talks about Brooks's Law, which states that adding people to an ongoing software project will be delayed, not hastily. We will understand how the strategy design pattern works and analyze the real problem and how to solve it using this design pattern.*
Key words: *Design patterns, productivity, efficiency, performance, application quality, strategy pattern.*

JEL CLASSIFICATION: C60, C61, M11

ВВЕДЕНИЕ

Шаблоны проектирования — это типичные решения проблем, которые часто возникают при разработке программного обеспечения. Они похожи на готовые планы, которые мы можем настроить для решения повторяющейся проблемы проектирования в нашем коде. Мы не можем просто найти шаблон и скопировать его в программу, как мы можем, с доступными функциями или библиотеками. Модель — это не конкретный фрагмент кода, а общая концепция решения определенной проблемы. Мы можем проследить детали модели и реализовать решение, соответствующее реалиям нашей собственной программы.

Целью данного исследования - познакомится с паттернами проектирования, понять важность применения их в реальных проектах и изучить влияние шаблонов проектирования на качество приложений, особенно на эффективность их работы.

ОСНОВНОЕ СОДЕРЖАНИЕ

Чтобы ответить более подробно на этот вопрос, нужно хорошо изучить данную проблему. Узнать какие виды паттернов существует на данный момент, как они классифицируются, как мы можем их применить в проектах и как мы можем увидеть результат.

Анализ литературных источников

Мы можем вполне успешно работать, не зная ни одного паттерна. Более того, мы могли уже не раз реализовать какой-то из паттернов, даже не подозревая об этом. Важным аспектом является то, что нужно очень глубоко понимать все принципы и концепции объектно-ориентированного программирования для того, чтобы правильно применять шаблоны проектирования. Без данных знаний, мы лишь усложним работу и сильно повлияем на производительность и архитектуру проекта в целом.

В источнике [1] приведены три пункта почему важно знать шаблоны проектирования:

- Проверенные решения. Мы тратим меньше времени, используя готовые решения, вместо повторного изобретения велосипеда. До некоторых решений мы могли бы додуматься и сами, но многие могут быть для нас открытием.
- Стандартизация кода. Мы делаем меньше просчётов при проектировании, используя типовые унифицированные решения, так как все скрытые проблемы в них уже давно найдены.

- Общий программистский словарь. Мы произносим название паттерна, вместо того, чтобы час объяснять другим программистам, какой хороший дизайн мы придумали и какие классы для этого нужны.

Из-за того, что наш код становится стандартизированным, он автоматически становится и намного “чище” – чем “чище” наш код, тем продуктивность работников на проекте сильно увеличивается, качество продукта на высоком уровне и проект может продолжать свое существование на многие годы вперед, потому что архитектура проекта была качественно спроектирована.

В своей книге [2] “Чистый код”, Роберт Мартин говорит, что: “ По мере нарастания плохого кода продуктивность команды продолжает снижаться, асимптотически приближаясь к нулю. По мере снижения производительности руководство делает единственное, что может; добавляет в проект больше сотрудников в надежде на повышение производительности. Но новый персонал не разбирается в системном дизайне.

В своей книге Фредерик Брукс [3] говорит, что добавление рабочей силы к позднему проекту программного обеспечения делает его позже. Во-первых, требуется время пока коллеги начнут понимать, как работает проект, чтоб стать более продуктивными. Во-вторых, рост команды затрудняет общение между коллегами. В-третьих, некоторые действия являются последовательными, их нельзя выполнять параллельно, отсюда делаем вывод, что очень важно изначально сделать правильную архитектуру проекта и соблюдать чистоту в коде.

В источнике «Приёмы объектно-ориентированного проектирования, паттерны проектирования» [4] говорится о том, что шаблоны проектирования упрощают разработку программного обеспечения, упрощают повторное использование успешных проектов и архитектур. Выражение проверенных методов в виде шаблонов проектирования делает их более доступными для разработчиков новых систем. Шаблоны проектирования помогают нам выбрать альтернативы дизайна, которые делают систему многоразовой, и избегать альтернатив, которые ставят под угрозу возможность повторного использования. Шаблоны проектирования могут даже улучшить документацию и обслуживание существующих систем, предоставляя явную спецификацию взаимодействий классов и объектов и их основного предназначения. Проще говоря, шаблоны проектирования помогают дизайнеру быстрее получить «правильный» дизайн.

Описание использованного метода исследования.

На данный момент я работаю над проектом, который включает в себя очень сложную логику сканирования продуктов в супермаркетах, о которой я буду говорить более подробно ниже. Методология, использованная в исследовании, является «прикладным научным исследованием», которое представляет собой оригинальную исследовательскую деятельность для накопления новых знаний, но ориентированную, прежде всего, к конкретной практической цели.

Результаты. Я провел исследование исходя из читаемой литературы и собственного и разобрался как работает паттерн проектирования Стратегия.

Существует 3 классификации шаблонов – порождающие, структурные и поведенческие.

Шаблон “Стратегия” является поведенческим паттерном проектирования – это означает, что данный шаблон проектирования определяет алгоритмы и способы реализации взаимодействия различных объектов и классов.

По определению в книге [4] Стратегия – это поведенческий паттерн проектирования, который определяет семейство схожих алгоритмов и инкапсулирует каждый из них в собственный класс, после чего алгоритмы можно взаимозаменять прямо во время исполнения программы.

Проблема, с которой я столкнулся заключается в том, что в логике мы имели очень много условных утверждений со схожей системой проверки, которые указывают нам, что нужно применять данный паттерн проектирования. Как и говорилось ранее, мы работаем над технологией самостоятельного сканирования продуктов в магазинах пользователем.

Она включает в себе режим офлайн и онлайн сканирования продуктов в магазинах. К этому каждый раз когда пользователь будет заходить в приложении начнутся реализовывать процессы проверки:

1. Если пользователь имеет доступ к интернету или нет.
2. Если пользователь включил свою геолокацию.
3. Если пользователь имеет карту магазина, ассоциированную с его профилем.
4. Если пользователь находится в магазине.
5. Если магазин, в котором находится пользователь имеет возможность для самостоятельного сканирования продуктов покупателем.
6. Если пользователь проверил все согласия пользования.

И много еще разных проверок. А что, если в будущем понадобится поменять их местами? Или понадобиться добавить новые проверки? Или поменять некоторые проверки местами?

Чтобы в будущем решать данные проблемы за короткое время и без больших затрат, мы решили, что лучше всего в нашем случае будет использовать паттерн проектирования “Стратегия”. В качестве контекста мы создали конкретный класс “SelfScanningCheckContext”, который хранит ссылку на объект конкретной стратегии, работая с ним через общий интерфейс стратегий. В качестве Стратегии мы создали интерфейс, который называется “SelfScanningCheckStrategy”, он определяет интерфейс, общий для всех вариаций алгоритма. Контекст использует этот интерфейс для вызова алгоритма. Для контекста неважно, какая именно вариация алгоритма будет выбрана, так как все они имеют одинаковый интерфейс. Конкретные стратегии для каждой задачи имеют названия: CheckIfInternetConnectionIsOn, CheckGeolocation, CheckIfUserIsInStore, CheckIfUserHasACard, CheckIfStoreAllowSelfScanning, CheckIfConsentHasBeenGiven и так далее. Данные стратегии реализуют различные вариации алгоритма.

Во время выполнения программы контекст получает вызовы от клиента и делегирует их объекту конкретной стратегии. Клиент должен создать объект конкретной стратегии и передать его в конструктор контекста. Кроме этого, контекст должен иметь возможность заменить стратегию на лету, используя сеттер. Благодаря этому, контекст не будет знать о том, какая именно стратегия сейчас выбрана. В результате мы получили гибкую архитектуру в данной части целой архитектуры, которая с легкостью в будущем может быть изменена без дополнительных затрат как денежных, так и временных.

ВЫВОДЫ

Вот, собственно, и весь смысл использования паттернов проектирования. Как мы видим, имеется схожесть паттернов проектирования с ситуациями из реальной жизни. Они существенно упрощают разработку программного обеспечения, делая ее более гибкой для изменения и добавления нового функционала, дает возможность эффективно решать часто встречающиеся проблемы.

Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем. Шаблон даёт решению своё имя, что облегчает коммуникацию между разработчиками, позволяя ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация деталей решений: модулей, элементов проекта, — снижается количество ошибок. Применение шаблонов концептуально сродни использованию готовых библиотек кода. Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова. Набор шаблонов помогает разработчику выбрать возможный, наиболее подходящий вариант проектирования [5].

БИБЛИОГРАФИЯ:

1. https://refactoring.guru/ru/design-patterns/why-learn-patterns
2. Martin, Robert C. Clean Code: *A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ: Prentice Hall, 2009.

3. Brooks, Frederick P., Jr., 1931-. The Mythical Man-Month: *Essays on Software Engineering*. Reading, Mass.: Addison-Wesley Pub. Co., 1982.
4. Design Patterns, *Elements of Reusable Object-Oriented Software*. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Published October 1994.
5. https://ru.wikipedia.org/wiki/Шаблон_проектирования#Плюсы.

Coordonator ştiinţific: CAPAŢINA Valentina, dr., conf. univ.
Academia de Studii Economice din Moldova
Republica Moldova, Chișinău, str. Bănulescu-Bodoni 61, www.ase.md
e-mail: valentina@ase.md