Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Архитектура программных систем»

**Отчет**

По лабораторной работе №2

Выполнил:

*Митрофанов Е. Ю.*

*Р33101*

Преподаватель:

*Перл И. А.*

Санкт-Петербург, 2021 г.

Задание:

Из списка шаблонов проектирования GoF и GRASP выбрать 3–4 шаблона и для каждого из них придумать 2–3 сценария, для решения которых могут быть применены выбранные шаблоны.

Сделать предположение о возможных ограничениях, к которым можем привести использование шаблона в каждом описанном случае. Обязательно выбрать шаблоны из обоих списков.

GoF шаблон “Прототип” (Prototype)

**Сценарии использования**

* Spawn-er в гейм-дизайне

Представим ситуацию, когда игровое окружение процедурно генерируется. Это может быть сделано для увеличения размера уровня или для разнообразия. При процедурное генерации использоваться так называемый «спавнер» почти всегда необходимо. В этом и помогает шаблон проектирования «прототип». Допустим у нас есть город, в котором существуют жители. У них есть простой искусственный интеллект, который нужен только для того, чтобы была иллюзия живого мира. Прописывать поведения каждому невозможно, в такой ситуации создается оригинальный, изначальный прототип, которому задается поведение, и этот объект просто копируется, ведь разнообразие в данном случае не так важно.

* Автоматизация создания базы данных производителя

Представим производство по сборке, к примеру, моторов для Rolls Royce. Для создания нового двигателя конструктор получает прототип двигателя для разных типов машин и просто меняет серийный номер. Не нужно будет создавать новый подкласс каждый раз, а при изменении типов двигателя нужно просто передать другие объекты. При необходимости нужно будет переписывать всего несколько методов, когда остальные остаются без изменений.

**Ограничения:**

Такой подход накладывает серьезные ограничения – каждый объект должен реализовывать операцию клонирования, но при необходимости глубокого клонирования, когда используются ссылки, указатели, задача такого клонирования сильно усложняется. К первому примеру выше, если мы хотим в будущем поменять поведения только некоторых объектов, задача станет намного сложнее.

GoF шаблон “Состояние” (State)

**Сценарии использования**

* Инструмент проверки и рецензирования научных статей

Возьмем пример сайт для публикации научных статей и исследований. Процесс создания статьи можно разделить на несколько состояний – публикация на рассмотрении, на модерации и рецензировании, и готова к публикации для всех. Во время каждого из состояний инструменты редактирования, просмотра и другие функции отличаются. Паттерн «состояние» позволит не создавать отдельные разделы или страницы сайта под каждый функционал, а менять его в зависимости от состояния.

* Приложение музыкального сервиса – плеера

Рассмотрим приложение Spotify для прослушивания музыки. Здесь мы так же можем выделить несколько состояний прослушивания. Просушивание альбома, своего плейлиста, подкаста или радио. В каждом случае функционал будет частично отличаться. Для своих плейлистов нужны функции рекомендаций и авто воспроизведения новых треков, для радио требуется возможность выбора жанра или исполнителя, для подкастов нужны комментарии или оценки. Но в то же время все состояния имеют общий базовый функционал паузы, уровня громкости и т. д.

* Приложение YouTube для телевизора

YouTube имеет возможность отправить выбранное видео на телевизор, где так же установлено приложение. В этим случает смартфон становится «пультом управления» воспроизведения видео. Остается весь функционал приложения, включая очередь воспроизведения, лайки и громкость, хотя само видео показывается на другом экране. Здесь явно видно разделение на два разных состояния внутри одного приложения.

**Ограничения:**

С одной стороны, паттерн освобождает нас от большого количество условных операторов state-машины, но помимо этого происходит сильное усложнение и уменьшение гибкости кода, особенно если самих состояний мало и меняются они редко.

GRASP шаблон “Устойчивый к изменениям” (Protected Variations)

Один из самых важных показателей хорошего кода – простота изменений и расширений функционала. Паттерн предполагает нахождений «неустойчивых» мест – фрагменты кода, которые чаще всего редактируются при новых изменениях, и устранения их, путем создания, к примеру, интерфейсов вокруг них.

Привести конкретный пример сложно, так как паттерн касается любого написания кода, но представим приложение, реализующее оплату по банковской карте. Реализована платежная система, передача банковский данных и защита данных. При портировании приложения на Android и IOS помимо оплаты по банковской карте нужны функции Google Pay и Apple Pay. Придется сильно переписывать код, чтобы добавить новый функционал. Вместо этого можно разделить общую реализацию оплаты на составляющие, чтобы выбор способа оплаты не менял реализацию полностью.

**Ограничения:**

Иногда полностью изменить модель невозможно, изменения все равно будут. Кроме того, предугадать изменения тоже сложно, на это влияют различные особенности и способы взаимодействия частей приложения. Для хорошей реализации шаблона и написания качественного, соответствующего ему кода нежно много усилий и времени, необходимо учитывать множество факторов как фреймворки, платформы и различный особенности используемого языка.

Выводы:

В ходе выполнения лабораторной работы я глубже изучил паттерны программирования GoF и GRASP. После погружения в них я пришел к следующим выводам: шаблоны в первую очередь предназначены не только для оптимизации написания кода, но и для того, чтобы разработчикам было легче понимать чужой код. Разложив что-то на составляющие, легче понять общую картину. С другой стороны, эффективность использования паттернов в некоторых ситуациях сходит на нет из-за сильного усложнения кода и процесса разработки даже при работе над системами малых масштабов.