Антипаттерн и паттерны проектирования

Антипаттерн - подход к решению задач, который является неэффективным. Рассмотрение антипаттерна включает в себя не только неправильно решение проблемы, но и нахождение правильного решения.

Противоположным подходом к решению задач, наиболее эффективными решениями, называется - паттернт. Хорошей практикой программирования является избегания антипаттернов. Использование антипаттернов влечет негативные последствия.

Разивитие программирования и увеличение количества создаваемых программ привело к появлению различных однотипных и схожих проблем возникающих в процессе разработки. Это привело к появлению шаблонов проектирования (design patterns) - изученных и проверенных на практике способов решения однотипных проблем.

Однако даже использование паттернов проектирования, в непредназначенных для применения ситуациях, пораждает появление еще больших проблем, чем до их использования. Ситуация обусловлена тем, что у начинающих программистов, как и у работников любой другой сферы, изначально недостаточно базы знаний и опыта работы. От сюда появление проблем вытикает их неспособности в сжатые сроки создать программу, что ведет к неэффективной разработке и некачественно написанному программному коду. Казалось бы, все нормально, программа работает, но при добавлении какого-либо нового функционала мы упремся в какое-то ограничение.

Использование паттернов проектирования начинается не только с момента продумываения архитектуры программы, но и применяется при последующей подержке программы на всем этапе ее жизненного цикла. Например, в процессе поддержки программы выявляется проблема, допущенная на начальном этапе и верным решением исправления проблемы будет использования одного из подходящих для этой ситуации паттерна проектирования.

Рассмотрение антипаттерна принято разделять на 3 типа:

- архитектура (антипаттерны архитектора)
- разработка (антипаттерны разработчика)
- руководство (антипаттерны менеджера)

Блоги

Информационные технологии
Проектирование и архитектура ПО
Мысли и размышления
Интервью
Маркетинг и продажи
Менеджмент
Информационная безопасность
Программирование
История ИТ
Интернет
Стартапы
Исследования и прогнозы в IT

Архитектурные антипаттерны

Архитектурные антипаттерны (architectural antipatterns) - это проблемы, возникающие в результате неправильно принятых решений архитекторов программного обеспечения. Структура программы, взаимосвязь ее внутренних компонентов продумана крайне не эффективно, либо вообще не продумана. От сюда при разработке будут возникать множество вопросов по архитектуре того или иного компонента программы.

Типичные проблемы, связанные с архитектурой системы:

- **Инверсия абстракции (Abstraction inversion)** - сокрытие части функциональности от внешнего использования, с расчетом на то, что никто не будет его использовать
- **Неопределённая точка зрения (Ambiguous viewpoint)** - представление модели без спецификации её точки рассмотрения
- **Большой комок грязи (Big ball of mud)** - программа с нераспознаваемой структурой
- **Бензиновая фабрика (Gas factory)** - необязательная сложность дизайна
- **Затычка на ввод данных (Input kludge)** - забывчивость в спецификации и выполнении поддержки неверного ввода
- **Раздувание интерфейса (Interface bloat)** - разработка интерфейса очень мощным и очень сложным для реализации
- **Волшебная кнопка (Magic pushbutton)** - выполнение результатов действий пользователя в виде неподходящего интерфейса
- **Перестыковка (Re-Coupling)** - процесс внедрения ненужной зависимости
- **Дымоход (Stovepipe System)** - редко поддерживаемая сборка плохо связанных компонентов
- **Состояние гонки (Race hazard, Race condition)** - непредвидение возможности наступления событий в не ожидаемом порядке
- **Мышиная возня** - создание множества мелких и абстрактных классов для решения одной конкретной задачи более высокого уровня
- **Членовредительство (Mutilation)** - излишнее «затачивание» объекта под определенную очень узкую задачу так, что объект не способен будет работать с другими схожими задачами
- **Сохранение или смерть (Save or die)** - сохранение изменений в конфигурации на жесткий диск только при завершении приложения

Антипаттерны разработки

Антипаттерны разработки (development antipatterns) - это проблемы, возникающие в результате неправильно принятых решений разработчиков программного обеспечения. Даже если архитектура программы очень хорошо продумана, то при ее реализации могут возникнуть некоторые вопросы, неправильное решение которых способно привести к некачественно созданной программе.

Рассмотрим антипаттерны разработки. Проблем, с которыми сталкиваются программисты, может быть большое множество.

Антипаттерны в объектно-ориентированном программировании

- **Базовый класс-утилита (BaseBean)** - наследование функциональности из класса, вместо делегирования к нему
- **Анемическая модель домена (Anemic Domain Model)** - боязнь размещения логики в объектах предметной области
- **Вызов предка (Call super)** - для добавления функциональности методу класса-потомка нужно вызвать эти же методы класса-предка

- **Ошибка пустого подкласса (Empty subclass failure)** - создание класса не проходящего пустого класса из-за того, что у наследуемого от него другого класса другое поведение
- **Божественный объект (God object)** - большое количество методов в одном классе
- **Объектная клоака (Object cesspool)** - переиспользование объектов, в непригодном для переиспользования состоянии
- **Полтергейст (Poltergeist)** - объекты, передающие данные другим объектам
- **Проблема йо-йо (Yo-yo problem)** - размытость сильно связанного кода по иерархии классов (например, кода, выполняемого по порядку)
- **Одиночество (Singletonitis)** - неуместное использование паттерна одиночка
- **Приватизация (Privatisation)**- сокрытие функциональности в приватной секции (private), затрудняющее его расширение в классах-потомках
- **Френд-зона (Friend zone)** - неуместное использование дружественных классов и дружественных функций (в языке программирования C++)
- **Каша из интерфейсов (Interface soup)** - объединение нескольких интерфейсов, разделенных в один, по принципу изоляции интерфейсов (Interface segregation)
- **Висящие концы** - интерфейс, методы которого бессмысленны и реализуются «пустышками»
- **Заглушка (Stub)** - использование в объекте уже имеющийся малоподходящий по смыслу интерфейс, вместо создания нового

Антипаттерны в кодировании

- **Ненужная сложность (Accidental complexity)** - ненужная сложности в решении
- **Действие на расстоянии (Action at a distance)** - взаимодействие между широко разделёнными частями программы
- **Накопить и запустить (Accumulate and fire)** - установка параметров подпрограмм в глобальных переменных
- **Слепая вера (Blind faith)** - недостаточная проверка исправления ошибки или правильной работы подпрограммы
- **Лодочный якорь (Boat anchor)** - сохранение более не используемой части системы
- **Активное ожидание (Busy spin, busy waiting)** - потребление ресурсов процессорного времени во время ожидания события, при помощи постоянно повторяемой проверки, вместо того, чтобы использовать асинхронное программирование
- **Кэширование ошибки (Caching failure)** - забывать сбросить флаг ошибки после её обработки
- **Воняющий подгузник (The Diaper Pattern Stinks)** - сброс флага ошибки без её обработки или передачи вышестоящему обработчику ошибки
- **Проверка типа вместо интерфейса (Checking type instead of membership, Checking type instead of interface)** - проверка типа объекта, когда требуется только определённый интерфейс
- **Инерция кода (Code momentum)** - сверхограничение части программы через использование ее поведения в других частях программы
- **Кодирование путём исключения (Coding by exception)** - добавление нового кода для поддержки каждого распознанного случая
- **Таинственный код (Cryptic code)** - использование аббревиатур вместо понятных имен
- **Жёсткое кодирование (Hard code)** - внедрение предположений об окружении программы в большом количестве точек её реализации
- **Мягкое кодирование (Soft code)** - боязнь жёсткого кодирования, приводящая к настраиванию абсолютно всего.
- **Поток лавы (Lava flow)** - сохранение лишнего или низкокачественного кода, потому что его удаление слишком дорого или имеет непредсказуемые последствия

- **Волшебные числа (Magic numbers)** - использование числовых констант без объяснения их смысла
- **Процедурный код (Procedural code)** - когда другая парадигма является более подходящей
- **Спагетти-код (Spaghetti code)** - код, запутанный как макароны.
- **Лазанья-код (Lasagnia code)** - использование большого количества уровней абстракции
- **Равиоли-код (Ravioli code)** - объекты склеены между собой как пельмени, что не позволяет делать рефакторинг
- **Мыльный пузырь (Soap bubble)** - объект, инициализированый мусором, и притворяющийся, что содержит данные
- **Мьютексный ад (Mutex hell)** - слишком большое количество объектов синхронизации между потоками
- **(Мета-)шаблонный рак (Template cancer)** - использование шаблонов там, где их использование не оправдано

Методологические антипаттерны

- **Программирование методом копирования-вставки (Copy and paste programming)** - копирование и использование кусков уже имеющегося кода.
- **Дефакторинг (De-Factoring)** - уничтожение функциональности и замены её документацией
- **Золотой молоток (Golden hammer)** - огромная уверенность, что любимое решение универсально и его можно использовать везде.
- **Фактор невероятности (Improbability factor)** - предположение о невозможности того, что сработает известная ошибка
- **Преждевременная оптимизация (Premature optimization)** - оптимизация кода когда это пока не нужно делать
- **Программирование методом подбора (Programming by permutation)** - подход к программированию небольшими изменениями
- **Изобретение колеса/велосипеда (Reinventing the wheel)** - создание решение с нуля, а не использование готового решения
- **Изобретение квадратного колеса (Reinventing the square wheel)** - создание плохого решения, когда есть хорошее и готовое решение
- **Самоуничтожение (Self-destruction)** - фатальная ошибка или нестандартное поведение программы, приводящая к отказу в обслуживании, возникшая из-за менее серьёзной ошибки.
- **Два тоннеля (Two tunnel)** - добавление новой функциональности в отдельное приложение вместо расширения имеющегося.
- **Коммит-убийца (Commit assasin)** - внесение изменений в систему контроля версий без проверки влияния на другие части программы.
- Антипаттерны управления конфигурацией
- **Ад зависимостей (Dependency hell)** - разрастание программных продуктов и библиотек, приводящее к сложности установки новых и удаления старых.

Разные

Дым и зеркала (Smoke and mirrors) - демонстрация, как будут выглядеть ненаписанные функции
Раздувание ПО (Software bloat) - последующие версии системы требовать всё больше и больше ресурсов
Функции для галочки - добавление в программу плохо реализованных и не связанных между собой функций (обычно, для пиара и рекламы)

Организационные антипаттерны

Организационные антипаттерны (managerial antipatterns) - это проблемы, возникающие в результате неправильно принятых решений руководителем проекта - менеджером. Менеджер проекта отвечает за взаимодействие между собой всех кто занимается разработкой программы. Напрямую от него зависит принятие решений по срокам проекта и выделяемым ресурсам на его разработку, что влияет на правильность работы всего проекта.

Проблемы, с которыми сталкиваются менеджеры:

- **Аналитический паралич (Analysis paralysis)** - большие затраты на анализ и проектирование, что приводит к закрытию проекта до начала его реализации
- **Дойная корова (Cash cow)** - в продукт, приносящий выгоду без вложений не вкладываются средства в развитие и разработку новых продуктов
- **Продолжительное устаревание (Continuous obsolescence)** - выделение больших усилий на портирование программы в новые окружения
- **Сваливание расходов (Cost migration)** - перенос расходов на проект к уязвимому отделу или бизнес-партнёру
- **Раздутый улучшизм (Creeping featurism)** - добавление новых улучшений в ущерб суммарному качеству программы
- **Раздутый элегантизм (Creeping elegance)** - непропорциональное улучшение красивости кода в ущерб функциональности и качеству системы
- **Разработка комитетом (Design by committee)** - разработка проекта без централизованного управления, либо при некомпетентном руководстве
- **Неуёмная преданность (Escalation of commitment)** - продолжение реализации решения после того, как доказана его ошибочность
- **Я тебе это говорил (I told you so)** - игнорирование мнения профессионала
- **Управление основанное на числах (Management by numbers)** - излишнее внимание к численным показателям, имеющим не очень важное значение
- **Драконовские меры (Management by perkele)** - неоправданно жесткий стиль управления
- **Управление грибами (Mushroom management)** - недостаточное информирование работников о выполняемой работе
- **Расползание рамок (Scope creep)** - потеря контроля над разрастанием проекта
- **Привязка к поставщику (Vendor lock-in)** - жёсткая привязка к поставщику
- **Тёплое тело (Warm Bodies)** - человек, чей вклад в проект под сомнением
- **Единственный знающий человек (Single head of knowledge, SHOK)** - важными сведениями или навыками обладает один человек в проекте, уход которого негативно сказывается на проекте
- **Рыцарь на белом коне (Knight in shining armor, KISA)** - человек, который пытается починить всё, при этом не сообщая никому, что именно он сделал

Как часто и какие именно антипаттерны вам доводилось применять при разработке проектов?

2018-04-19 **Программирование**

Автор

Илья

Программист с образованием в области IT и опытом разработки на разных языках. Автор статей по программированию. Общий опыт работы в сфере IT и интернета более 5 лет.

0 комментариев

Чтобы оставить комментарий зарегистрируйтесь, или войдите в аккаунт.