Шаблоны (или *паттерны*) *проектирования* (design patterns) – это многократно используемые решения распространенных проблем, возникающих при разработке программного обеспечения[1]. По мере накопления личного опыта и профессионального роста программист обычно замечает сходство новых проблем проектирования с решёнными им ранее. В дальнейшем приходит осознание того, что решения похожих проблем представляют собой повторяющиеся шаблоны. Зная эти шаблоны, опытные программисты распознают ситуацию их применения и сразу используют готовое решение, не тратя времени на предварительный анализ проблемы.

Главная польза каждого отдельного шаблона состоит в том, что он описывает решение целого класса абстрактных проблем. Тот факт, что каждый шаблон имеет свое имя, облегчает дискуссию между разработчиками. Таким образом, за счёт шаблонов производится унификация терминологии, названий модулей и элементов проекта. Однако есть мнение, что слепое применение шаблонов из справочника, без осмысления причин и предпосылок выделения шаблона, замедляет профессиональный рост программиста, так как подменяет творческую работу механической подстановкой. Люди, придерживающиеся данного мнения, считают, что знакомиться со списками шаблонов следует тогда, когда программист «дорос» до них в профессиональном плане. Хороший критерий нужной степени профессионализма - выделение шаблонов самостоятельно, на основании собственного опыта. При этом, разумеется, знакомство с теорией, связанной с шаблонами, полезно на любом уровне профессионализма и направляет развитие программиста в правильную сторону.

Структурные шаблоны – показывают, как объекты и классы объединяются для образования сложных структур.

Порождающие шаблоны – управляют и контролируют процесс создания и жизненный цикл объектов.

Шаблоны поведения – используются для организации, управления и объединения различных вариантов поведения объектов.

Антипаттерны (anti-patterns), также известные как *ловушки* (pitfalls) - это классы наиболее часто внедряемых плохих решений проблем. Они изучаются в случае, когда их хотят избежать в будущем, и некоторые отдельные случаи их могут быть распознаны при изучении неработающих систем.

- *Ненужная сложность* (Accidental complexity) - внесение ненужной сложности в решение. Ненужная сложность не обусловлена действительной сложностью решаемой проблемы, а искусственно внесена при разработке архитектуры и дизайна ПО.
- *Слепая вера* (Blind faith) - недостаточная проверка корректности исправления ошибки или результата работы подпрограммы. Альтернативой антипаттерну является разработка через тестирование и, в частности, модульные тесты.
- *Лодочный якорь* (Boat anchor) - сохранение более не используемой части системы. Чтобы уменьшить влияние данного антипатерна, рекомендуется устаревшие компоненты системы выносить в отдельные библиотеки, дабы не захламлять основной набор классов и методов.
- *Кэширование ошибки* (Caching failure) - забывать сбросить флаг ошибки после её обработки. Согласно своему названию, антипаттерн обычно возникает в системах с кэшированием. Один из способов устранения – полная очистка кэша в случае возникновения ошибки.
- *Жёсткое кодирование* (Hard code) - внедрение предположений об окружении системы в слишком большом количестве точек её реализации. Примером является явная запись имени и пути к файлу в коде. Для устранения антипаттерна следует вместо явных значений использовать именованные константы или параметры из конфигурационных файлов.
- *Мягкое кодирование* (Soft code) - патологическая боязнь жёсткого кодирования, приводящая к тому, что настраивается всё что угодно, при этом конфигурирование системы само по себе превращается в программирование.
- *Поток лавы* (Lava flow) - сохранение нежелательного (излишнего или низкокачественного) кода по причине того, что его удаление слишком дорого или будет иметь непредсказуемые последствия.
- *Магические числа* (Magic numbers) - включение чисел в алгоритмы без объяснений. Для устранения антипаттерна следует использовать именованные константы, причём имя должно отражать смысл константы (см. антипаттерн Таинственный код).

[1] Алгоритмы не рассматриваются как шаблоны, так как они решают задачи вычисления, а не проектирования.

- *Процедурный код* (Procedural code) – ситуация, когда другая парадигма программирования является более подходящей для решения задачи.
- *Спагетти-код* (Spaghetti code) - системы, чья структура редко понятна, особенно потому что структура кода используется неправильно. Антипаттерн обычно проявляется в методах большого размера (20 строк кода и более). Для устранения антипаттерна выполняют выделение фрагментов кода в отдельные функции.
- *Мыльный пузырь* (Soap bubble) - класс, инициализированный мусором, максимально долго притворяется, что содержит какие-то данные.

Антипаттерны в объектно-ориентированном программировании.

- *Базовый класс-утилита* (BaseBean) - наследование функциональности из класса-утилиты вместо делегирования к нему.
- *Вызов предка* (CallSuper) - для реализации прикладной функциональности методу класса-потомка требуется в обязательном порядке вызывать те же методы класса-предка. Вместо данного антипаттерна следует использовать паттерн Шаблонный метод.
- *Божественный объект* (God object) - концентрация слишком большого количества функций в одиночной части дизайна (классе). Любой класс должен иметь единственное назначение, которое можно описать несколькими словами. Большие классы следует разбить на мелкие, возможно с применением агрегирования, делегирования и наследования.

Антипаттерны в разработке ПО.

- *Большой комок грязи* (Big ball of mud) - система с нераспознаваемой структурой.
- *Бензиновая фабрика* (Gas factory) - необязательная сложность дизайна.
- *Затычка на ввод данных* (Input kludge) - забывчивость в спецификации и выполнении поддержки возможного неверного ввода. Антипаттерн устраняется продуманным алгоритмом проверки (валидации) пользовательского ввода.
- *Раздувание интерфейса* (Interface bloat) - изготовление интерфейса очень мощным и очень трудным для осуществления. Альтернативой этому антипаттерну служит использование таких шаблонов, как Адаптер или Посетитель.

Методологические антипаттерны.

- *Программирование методом копирования-вставки* (Copy and paste programming) - копирование (и лёгкая модификация) существующего кода вместо создания общих решений. Симптом этого антипаттерна: после внесения изменений программа в некоторых случаях ведёт себя также, как и раньше. Для устранения антипаттерна требуется выделить повторяющийся код в отдельный метод.
- *Дефакторинг* (De-Factoring) - процесс уничтожения функциональности и замены её документацией.
- *Золотой молоток* (Golden hammer) - сильная уверенность в том, что любимое решение универсально применимо. Название происходит от английской поговорки «когда в руках молоток, все проблемы кажутся гвоздями».
- *Фактор невероятности* (Improbability factor) - предположение о невозможности того, что сработает известная ошибка.