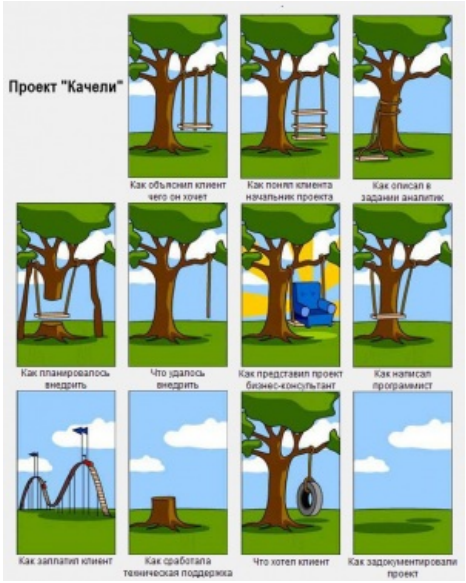
Анти-паттерн — Lurkmore

> **Эта статья написана интеллектуальным большинством.**
>
> 
>
> В итоге от её чтения ни пользы, ни удовольствия. Поскольку с самим большинством мы ничего сделать не в состоянии, хотя бы статью следует переписать или спрятать куда-нибудь подальше.

Анти-паттерн (от англ. *anti-pattern*; он же ловушка, англ. *pitfall*) — это то, как поступать не надо, но как всё равно все, всегда и повсюду поступают, сколько их ни предупреждают. Потому как дедлайны. Потому как бизнес. Потому как долбоёбы.

Термин пришел из информатики, из книги «Банды четырёх» «Шаблоны проектирования», которая заложила примеры практики хорошего программирования. Авторы назвали эти хорошие методы «шаблонами (паттернами) проектирования», а антиподами им и являются, соответственно, «анти-паттерны». В принципе, анти-паттерны применимы не только в ИТ, но и в любой другой области человеческой деятельности.



Пример активного применения анти-паттернов в проекте.

Анти-паттерны программирования

- **Жёсткий код**, от англ. *hard code* — про него ещё говорят «прибито гвоздями». Код настолько привязан к конкретной аппаратной конфигурации и/или системному окружению, что оторвать его для переноса в другое место можно только гвоздодёром.
- **Мягкий код**, от англ. *soft code* — почти то же, что мягкий стул, или, техническим языком, код, конфигурируемый настолько гибко (и запутано), что хочется, чтобы лучше уж это был *hard code*. Возникает вследствие выноса в файлы конфигурации программной логики, что в разумных количествах, к слову, бывает иногда оправданно.
- **Магические числа**, от англ. *magic numbers* — то, на чём нередко строится *hard code*. Всяческие константы в программе без пояснения их физического (или любого иного) смысла. Как правило, при изменении магического числа или его удалении код магически перестаёт работать вовсе. Типичные примеры — 17 и 54.
- **Магические строки**, от англ. *magic strings* — как магические числа, но только это строки.
- **Магическая кнопка**, от англ. *magic button* — весь код написан в обработчике нажатия кнопки. Этим страдают на всех языках с графическим редактором гуя — C#, делфи.
- **Гонки**, англ. *race condition*, *race hazard* — начинаются тогда, когда программист попадает в чудный мир параллельного программирования с его нитями, потоками и семафорами.
- **Ненужная сложность**, от англ. *accidental complexity* — делать сложно то, что можно сделать просто. Как правило, программисту хочется побыстрее применить всё, что он изучил (и воон ту новую библиотеку для обмена данными по протоколу XXX), а ещё заполучить по результатам проекта сразу много новых строчек в резюме. Если это желание побеждает доводы рассудка, то в проекте появляется *accidental complexity*. Также см. статьи Индусский код и KISS.
- **Спагетти** (рус. солянка) из кода, от англ. *code spaghetti* — это тот самый код, который вам однажды дают сопровождать и после пяти минут ~~причёсывания вставших дыбом волос~~ просмотра которого вы молча закрываете редактор и открываете броузер на хуехантере.ру в поисках вакансии в более другой фирме, где не практикуют такого жестокого обращения с программистами. Характеризуется тем, что в одном куске кода описывается куча перемешанных до полного непонимания сущностей, которые по-хорошему необходимо разделять. Например, HTML теги в PHP коде вместо выноса всей верстки в шаблоны.
- **Катамари** — внедрение новых возможностей исключительно в виде внешних костылей и подпорок вместо периодического пересмотра базового функционала. В итоге со временем код превращается в тугой комок какого-то нечта, в чём даже сами разработчики отказываются ковыряться. В итоге получаем следующую порцию костылей и подпорок. Может применяться как в отдельно взятом фрагменте кода, так и глобально во всём проекте.
- **Божественный Объект**, от англ. *God Object* — небольшая часть кода, где сконцентрировано всё. Буквально всё. Возможно, там даже прячется немаленьких размеров галактика. Весь прочий код программы — исключительно декорация вокруг Божественного Объекта.
- **Детонатор**, от англ. *detonator* — очень распространён, но редко обнаруживаем. Наглядный пример: использование в вычислениях с датами поля из двух цифр. Бомба заложена, и детонатор рано или поздно сработает

- **Не-виноватая-я**, англ. *absolver* — паттерн встречается в коде, написанном бывшими сотрудниками компании. В таком коде заключено столько старых проблем, что теперешние сотрудники могут защитить свои наработки от обвинений, утверждая, что именно чужой код — причина всех возникающих ошибок. Также известен под именем *Это-Не-Моя-Правка*.
- **Исторический вклад**, от англ. *stake* — паттерн имеет место в программе, написанной сотрудником, который впоследствии получил продвижение по службе. Несмотря на изобилие ошибок в программе, вклад этого сотрудника слишком велик, чтобы позволить кому-либо начать переписывать код, поскольку он является апофеозом технических достижений этого товарища.
- **Паблик Морозов** (сугубо русская идиома, нет английского эквивалента) — класс, который по запросу выдаёт доступ ко всем, даже приватным, свойствам класса-родителя. Не столько плохо само по себе, сколько означает, что в коде что-то не так.
- **Спасти рядового Райана**, от англ. *Private Ryan* — паттерн, в котором доступ к полю получить в принципе можно, но для этого надо снаряжать экспедицию, сопряженную со множеством опасностей.

Анти-паттерны системного администрирования

Специально для сисадминов есть два антипаттерна, в которые они попадают в зависимости от своей специализации.

- **Ад зависимостей**, от англ. *dependency hell* — для юниксовых сисадминов. Прямо проистекает из того факта, что строители юниксовых дистрибутивов (в первую очередь линупсов) однажды возгордились и решили ~~возвести Вавилонскую башню~~ упорядочить весь линуксовый софт, так, чтобы одна программа в дистрибутиве не мешала другой. Последствия этой попытки пользователи порой ощущают в виде «циклических зависимостей» и прочих дорожек в *dependency hell*.
- **Ад DLL**, от англ. *DLL hell* — персонально для сисадминов Windows. Проблема является частным случаем *ада зависимостей*, применительно к формату библиотек (DLL) в Windows, особенно в старых версиях этой системы.

Организационные анти-паттерны

Есть такое искусство — «пасти котов», или, скучно выражаясь, управление программистами. Рисование всяких там диаграммок Ганта и прочей псевдонаучной лабуды, которую потом все дружно игнорируют. Этим занимаются менеджеры. Не секрет, что в менеджеры уходят те, кто в ИТ пришёл к своему финишу, постарел и отупел, чтобы писать код и ваять вечное, а может быть, и вовсе кроме ворда и аутлука ничем не пользовался и потому подался сразу ~~в управдомы~~ управлять людьми. Поэтому антипаттерны в этой области просто-таки цветут пышным цветом. Дополнительную дезорганизацию в организацию разработки порой привносят сами разработчики, помогая менеджерам во внедрении анти-паттернов.

- **Управление грибами**, от англ. *mushroom management* — метод управления подчиненными по принципу: вы все муравьи, а я муравейник! Английский лозунг грибного менеджера — «we keep them in the dark and feed them a steady diet of manure» (Richard Henderson). Менеджер всячески ~~окучивает грибы~~ убеждает подчинённых в том, что кроме их непосредственных заданий, им не надо ничего знать, а утруждать мозги думами о проекте в целом — исключительно манагерский удел. Для внедрения данного метода следует разбить проект на несобираемые кусочки паззла, чтобы никто не понимал, что из них можно собрать вообще и затем выдавать каждому эти кусочки в достаточных количествах, чтобы грибы перманентно были заняты работой и не отвлекались на раздумья «а на уя вообще мы делаем именно так?». Как правило, грибной менеджмент приводит к полностью провальным финалам, поскольку менеджер сам забывает (либо изначально не понимает), к чему и зачем идёт проект. Что любопытно, грибной менеджмент практикуют не только менеджеры, но и отдельные программисты, очутившись на ролях старших в проекте. Как правило, это индикатор неквалифицированных программистов, боящихся раскрывать коллегам те немногие знания, которыми они обладают — чтобы не обнаружилось, насколько эти знания малы. Та же причина справедлива и для менеджеров.
- **Чайка-менеджер**, от англ. *seagull management* или *corporate seagull* — распространённый подкласс менеджеров, действующих в следующей последовательности: прилетел, поорал и улетел, оставив за собой кучки проектного дерьма, которые потом, матерясь, подчищают подчинённые. По поведению напоминают ~~морских воронов~~ чаек, парящих гордо между тучами и морем и прилетающих на берег с целью быстро пожрать и попутно опорожнить кишечники, за что они (менеджеры) и переняли данное наименование. Самый бестолковый для проектной деятельности тип, но начальство их любит, так как *корпоративные чайки* умеют громко ~~хлопать крыльями~~ заявлять о себе, выдавая копеечную деятельность за великие свершения. Живучи, их сложно поймать на ошибках, так как ошибок они не совершают (поскольку, как известно, для этого нужно хоть что-то делать).
- **Разработка комитетом**, от англ. *design by commitee* — аналог пословицы «у семи нянек дитя без глазу» применительно к ИТ. Раздувается в гигантские нежизнеспособные стандарты и спецификации с over 9000 уровней абстракции, которые потом всё равно никто не реализует. Примеры — модель OSI[*ЩИТО?*] и CORBA.
- **Аналитики-паралитики**, от англ. *analysis paralysis* — проект застревает на стадии анализа задачи, в бесконечных спорах «как делать лучше». В результате получается вообще никак.
- **Рыцарь на белом коне** (**РНБК**), от англ. *Knight in shining armor* (*KISA*) — непогрешимая личность, с

ЧСВ, равным как минимум квадрату собственной технической квалификации. Появляется на сцене в кризисный момент и начинает чинить всё подряд, как правило — чинит успешно, только без сообщений о том, какие изменения он сделал, и почему. Так что при очередном обновлении системы ничего не подозревающим сисадмином старые ошибки возвращаются на свои места, что даёт повод РНБК заявлять о тупости и некомпетентности окружающих, что ещё более увеличивает его ауру непогрешимости.
- **Аврал!**, от англ. *Tempest Method* — применяется не ранее, чем за несколько дней до сдачи проекта. При этом пограммисты начинают в срочном порядке генерировать код без комментариев и содержащий в себе связки бомб с *детонаторами*.
- **Охота на ведьм**, от англ. *witch hunt* — попытки отыскать козла отпущения после успешного внедрения анти-паттернов выше, но поскольку проблем это не решает, результатом будет являться отстрел всё новых и новых козлов отпущения. В финале игры должен оставаться только один — ~~Duncan McLeod~~ менеджер на белом коне, с подтверждениями некомпетентности предыдущей команды и уверениями руководства фирмы в том, что новая команда под его руководством достигнет небывалых проектных успехов.

Антихитрожопость

Как говорил рядовой программист Э. Дейкстра: «Компетентный программист полностью осознает строго ограниченные возможности своего черепа, поэтому подходит к задачам программирования со всей возможной скромностью».

Заблуждения по поводу потенциала мозга руководителя или программиста являются перманентным источником антипаттернов.

- **Раскрытие потенциала технологии на 110%**. Быстрое развитие платформ, средств разработки и их зарастание возможностями туманит мозги рядовому кодеру, который предполагает, что если он не будет писать код с десятью операторами на строку, использовать все возможные варианты генераторов, шаблонов, тегов, тонкие нюансы работы интерпретатора и низкосистемные хаки, то коллеги сочтут его полным junior'ом. Но, как правило, все получается ровно таки наоборот и каждая хитрожопость в коде с лихвой аукается при дальнейшей поддержке продукта, не говоря про то, что бедные разработчики платформ, сред и библиотек кучу фич включают просто для галочки, не уделяя им соответствующее внимание. При этом простых и понятных дедушкиных подходов, которые не обязательно означают квадратно-гнездовой подход, обычно с лихвой хватает для решения подавляющего большинства задач.

- **Манипулирование сроками**. Любому эффективному руководителю рано или поздно приходит в голову мысль, а не стоит ли назвать разработчикам срок в 3,14 раз меньше оценочного, чтобы пошевеливались, хотя, если инвестиции затуманили его мозг, то в 3,14 раза больше, чтобы в этот раз ну точно успеть к дедлайну. Мысль о том, что все будет относительно хорошо, только если план работ соотносится с оценкой объема работ, считается слишком банальной и не достойной настоящего гуру руководства. Как следствие — множество проектов заканчивается феерически.

- **Экономия**. Так как заказчик часто желает получить золотой замок за ведро пожухлого говна, то встает вопрос об экономии. Истинная хитрожопость диктует, что вместо того, чтобы сократить слой позолоты в два раза (обрезать фичи и требования), лучше сократить проектирование, прототипирование, внутренние конвенции, тесты, автотесты, команду разработчиков и прочие вещи, которые все равно не получается делать по-человечески. Разумеется, ответ на ключевой вопрос очевиден, так как на заводе для производства денатурата никак не удастся изготовить бочонок шотландского виски, что все вроде как головой понимают, но в душе все равно хотят.

- **Раскрытие кадрового потенциала на 110%**. В голову среднестатистического эффективного проджект-менеджера никак не укладывается, что программиста, написавшего в день десять строк, вполне можно назвать нормально поработавшим, а сто годных, отлаженных и протестированных строк — поработавшим отменно. Это приводит его к мысли, что в качестве программиста можно посадить секретаршу, так как у нее скорость набора выше, но так как секретарша отказывается, менеджерская логика подсказывает, что пора нагнетать панику, бегать со страпоном и всячески стимулировать процесс, дабы поиметь отдачу на 110%. В результате проект дохнет под собственным технологическом долгом (неудачные решения, неправильные оценки и тупо баги), не успев толком начаться. Правда, сам менеджер, как правило, двигает кони в первых рядах, а команда разработчиков, поправив резюме, на следующий же день переползает в какие-то другие конторы в надежде на лучшее.

Гиперссылки

- Большой неиллюстрированный каталог анти-паттернов (англ.)
- Проломы проектно-дизориентированного проектирования

Языки программирования
++i + ++i 1C AJAX BrainFuck C Sharp C++ Dummy mode Erlang Forth FUBAR
God is real, unless explicitly declared as integer GOTO Haskell Ifconfig Java JavaScript LISP
My other car Oracle Pascal Perl PHP Prolog Pure C Python RegExp Reverse Engineering
Ruby SAP SICP Tcl TeX Xyzzy Анти-паттерн Ассемблер Быдлокодер
Выстрелить себе в ногу Грязный хак Дискета ЕГГОГ Индусский код Инжалид дежице
Капча КОИ-8 Костыль Лог Метод научного тыка Очередь Помолясь Проблема 2000
Программист Процент эс Рекурсия Свистелки и перделки Спортивное программирование
СУБД Тестировщик Умение разбираться в чужом коде Фаза Луны Фортран Хакер
Языки программирования

w:Анти-паттерн