

# Анти-паттерны разработчика

DEV-014



LXFT  
LISTED  
NYSE

# YOUR TRAINER

## Expert IT Architect Viacheslav Kalashnikov



### PROFILE

- ◆ 25+ years of IT experience
- ◆ 15+ years of extensive experience in enterprise data warehousing, business intelligence and software architecture, including projects built from scratch as well as developed/enlarged
- ◆ proven ability to lead by example, consistently hit targets, improve best practices and organize time efficiently

### EXPERTISE:

- ◆ DataWarehousing
  - ◆ BigData
  - ◆ AWS
  - ◆ GCP
  - ◆ Oracle
  - ◆ Exadata
  - ◆ Teradata
  - ◆ Sybase ASE
  - ◆ Docker
  - ◆ Hadoop
  - ◆ MongoDB
  - ◆ Vertica
  - ◆ Sybase IQ
  - ◆ BusinessObjects
  - ◆ MicroStrategy
  - ◆ Informatica
  - ◆ Kubernetes/OpenShift
  - ◆ NiFi
  - ◆ WarehouseBuilder
  - ◆ DataStage
  - ◆ DataIntegrator
  - ◆ JMS, MQ, Kafka
  - ◆ RaspberryPi
  - ◆ Cognos
- ◆ Team leading skills, interviewing, mentorship, trainings in distributed teams.
  - ◆ Good communication skills: presentations, meetings, requirements gathering.

# Давайте знакомиться

- ◆ Как вас зовут?
- ◆ Почему вы пришли на тренинг?
- ◆ Какие ожидания от тренинга?



Вместо  
эпиграфа...



# Несколько вопросов

- ◆ **Что такое паттерн?**
- ◆ **Что такое анти-паттерн?**



# План тренинга

- ◆ Модуль 1. Введение (паттерны и антипаттерны)
- ◆ Модуль 2. Антипаттерны в архитектуре
- ◆ Модуль 3. Антипаттерны в разработке программного обеспечения
- ◆ Модуль 4. Антипаттерны в управлении разработкой ПО



# Введение (Паттерны и антипаттерны)

Модуль 1



# Антипаттерн это

- «**Отрицательное решение**» или решение, содержащее множество проблем для дальнейшего использования или поддержки проекта
- Представляет собой разрыв между архитектурной концепцией и реальном воплощении этой концепции
- Обеспечивает знания, которые позволяют предотвратить или исправить общие ошибки



# Почему мы изучаем антипаттерны

- Антипаттерны обеспечивают шаблон для идентификации известных проблем
- Антипаттерны обеспечивают обобщение реального мирового опыта в признании повторяющихся проблем в индустрии программного обеспечения
- Антипаттерны обеспечивают общий словарь для идентификации их проблем, а также для обсуждения способов их решения



# Откуда антипаттерны?

- ◆ Ускорение - Жесткие сроки часто приводят к игнорированию важных шагов



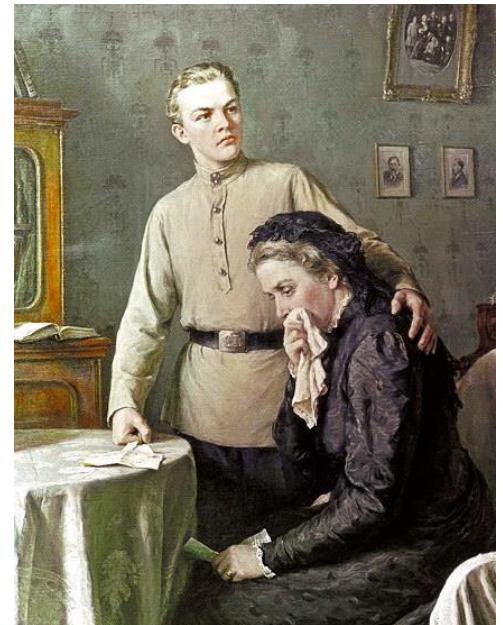
# Откуда антипаттерны?

- ◆ Апатия - Позиция не заботится о решении известных проблем



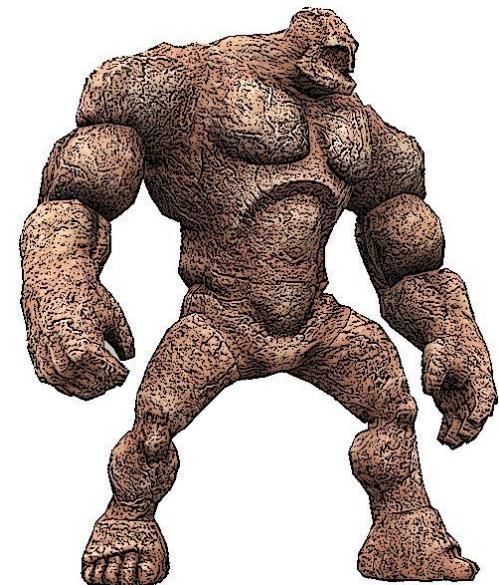
# Откуда антипаттерны?

- ◆ Узколобость - Отказ разработчиков использовать проверенные решения



# Откуда антипаттерны?

- ◆ **Лень - Принимаются наиболее простые решения**



# Откуда антипаттерны?

- ◆ Алчность - Жадность в создании системы может привести к очень сложному и трудноподдерживаемому программному обеспечению



# Откуда антипаттерны?

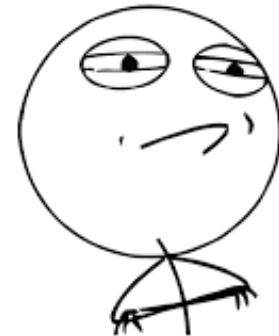
- ◆ **Невежество - Отсутствие или недостаточная мотивация, чтобы понимать очевидные вещи**



# Откуда антипаттерны?

- ◆ Гордость - Отказ использования существующих готовых решений, потому что они были изобретены не нами

**CHALLENGE ACCEPTED**

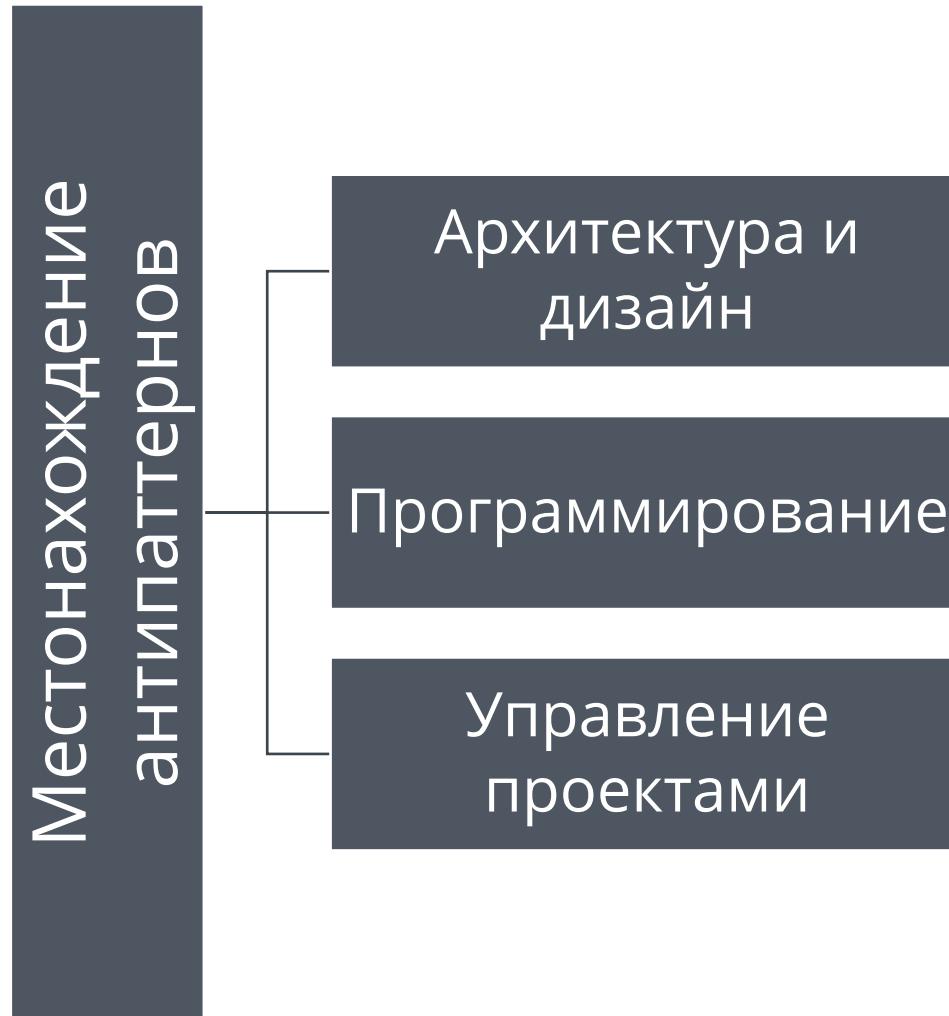


# Предварительные вопросы

- ◆ С какими часто встречающимися ошибками проектирования программного обеспечения вы сталкиваетесь?
- ◆ Каким образом вы стараетесь избежать этих ошибок?



# Где могут проявиться антипаттерны



# Описание антипаттерна

- ◆ **Имя**
  - Формальное имя антипаттерна
- ◆ **Альтернативные имена**
  - Другие имена, возможны юмористические («Паблик Морозов»)
- ◆ **Отношение к уровню дизайна**
  - На каком уровне дизайна следует рассматривать данный паттерн
- ◆ **Имя рефакторинга**
  - Имя решение, которое позволяет устраниить указанный антипаттерн

# Описание антипаттерна

- ◆ **Тип рефакторинга**
  - Идентифицирует тип рефакторинга для исправления ситуации. Рефакторинг может иметь отношение к программному обеспечению, технологиям, процессам или ролям
- ◆ **Главные причины**
  - Одна или несколько причин, приводящих к антипаттерну
- ◆ **Несбалансированные силы**
  - Определение первичной силы, которая игнорируется, или которой злоупотребляют в антипаттерне
- ◆ **Описание**
  - Общие фразы и юмористические анекдоты, описывающие проблему

# Описание антипаттерна

- ◆ **Справочная информация**

- Устанавливает предмет обсуждения антипаттерна

- ◆ **Общая форма**

- Общие характеристики антипаттерна, обзор природы проблемы

- ◆ **Симптомы и последствия**

- Список симптомов и связанных с ними последствий, вытекающих из антипаттерна

- ◆ **Типичные причины**

- Список уникальных причин в антипаттерне, которые должны быть связаны с соответствующими симптомами и последствия

# Описание антипаттерна

- ◆ **Известные исключения**

- Известные случаи, когда подобное поведение или процессы не являются антипаттерном

- ◆ **Описание рефакторинга**

- Детальное описание шагов по исправлению ситуации

- ◆ **Варианты**

- Другие известные варианты антипаттерна

# Описание антипаттерна

- ◆ **Пример**
  - Пример антипаттерна, базирующийся на реальном опыте
- ◆ **Связанные решения**
  - Список связей с другими антипаттернами и их решениями
- ◆ **Применение к другим точкам зрения и масштабу**
  - Описывает влияние антипаттерна на другие уровни масштаба дизайна

# Сокращенное описание антипаттерна

- ◆ **Исток**

- Ключевая причина, которая привела к появлению антипаттерна

- ◆ **Симптомы и последствия**

- Симптом наличия антипаттерна и последствия для проекта

- ◆ **Решение**

- Набор действий, позволяющий устраниить проблемы антипаттерна

# Типы антипаттернов

- ◆ Архитектурные антипаттерны
- ◆ Антипаттерны программирования
- ◆ Антипаттерны управления

# Антипаттерны в архитектуре программного обеспечения

Модуль 2



# Архитектурные антипаттерны

- ◆ Хорошая архитектура это критический фактор успеха в разработке программного обеспечения
- ◆ Архитектура ПО является подмножеством системной архитектуры, которая включает себя вопросы проектирования и разработки, выбор программно-аппаратной платформы
- ◆ Архитектурные антипаттерны фокусируются на уровне системы или предприятия



# Архитектурные антипаттерны

- ◆ Автогенерируемый дымоход (Autogenerated Stovepipe)
- ◆ Дымоход предприятия (Stovepipe Enterprise)
- ◆ Беспорядок (Jumble)
- ◆ Система дымоход (Stovepipe system)
- ◆ Обертка (Cover Your Assets)
- ◆ Зависимость от поставщика (Vendor Lock-In)
- ◆ Волчий билет (Wolf Ticket)
- ◆ Архитектура по наитию (Architecture Implication)
- ◆ Герои (Warm Bodies)



# Архитектурные антипаттерны

- ◆ **Дизайн комитетом (Design by Committee)**
- ◆ **Швейцарский нож (Swiss Army Knife)**
- ◆ **Изобретение колеса (Reinvent The Wheel)**
- ◆ **Абстракционизм (The Grand Old Duke Of York)**



# Автогенерируемый дымоход



# Автогенерируемый дымоход (Autogenerated Stovepipe)

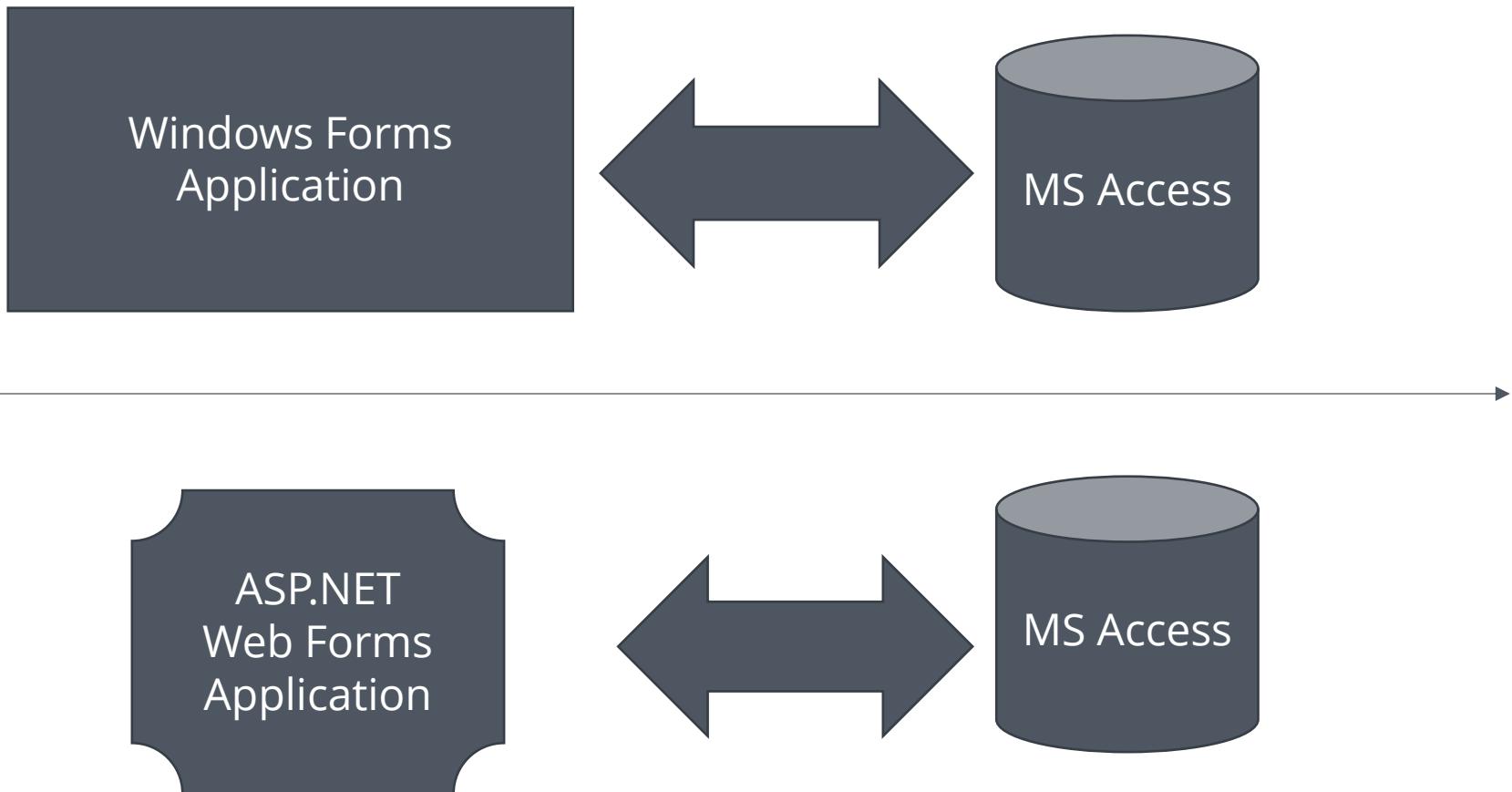
## ◆ Исток

- Конвертация существующего программного обеспечения «как есть» для работы в распределенных условиях  
(Сохраняется ранее определенный API без каких-либо изменений)

## ◆ Причины

- Разработчики не учитывают затраты на коммуникацию в распределенных системах
- Отсутствие желания что-либо менять в дизайне
- Сжатые сроки

# Автогенерируемый дымоход (Autogenerated Stovepipe). Пример



# Автогенерируемый дымоход (Autogenerated StovePipe)

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Автогенерируемый дымоход (Autogenerated StovePipe)

## ◆ Решение

- Необходимо полностью изменить интерфейс с учетом специфики распределенных приложений и снижении затрат на обмен сообщениями
- Используйте паттерны Remote Façade, Data Transfer Object
- Функциональность взаимодействия между подсистемами должна использовать новый интерфейс как единую точку входа



Это научный эксперимент

DEMOTIVATORS.RU

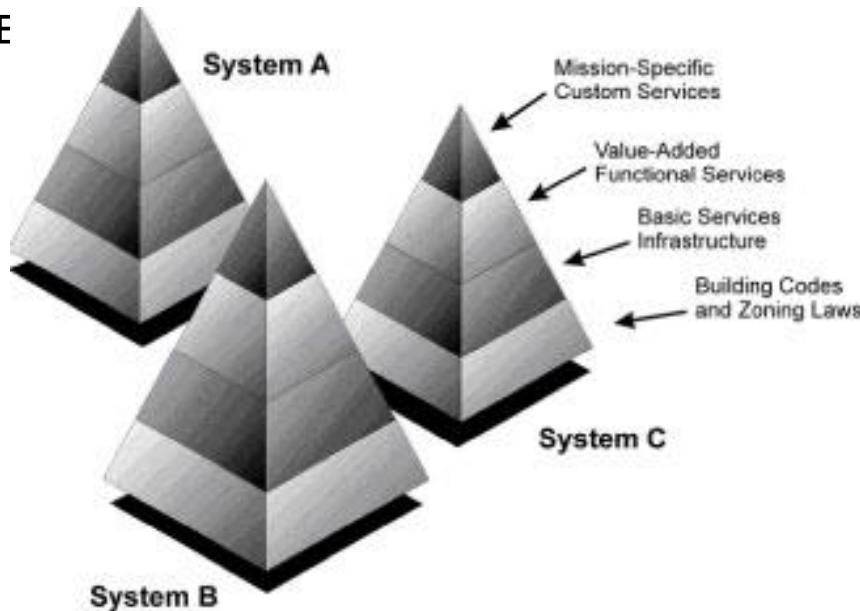
# Дымоход предприятия



# Дымоход предприятия (Stovepipe Enterprise)

## ◆ Исток

- Разрабатывается несколько независимых систем на предприятии, интеграция которых между собой резко повышает стоимость, предотвращает повторное использование, что приводит к сложностям в поддержке и обслужии



# Дымоход предприятия (Stovepipe Enterprise)

## ◆ Симптомы и последствия

- Несовместимая терминология, подходы и технологии между системами предприятия
- Хрупкость, монолитная архитектура и (или) недокументированная архитектура
- Неспособность к расширению с целью реагирования на требования бизнеса

# Дымоход предприятия (Stovepipe Enterprise)

## ◆ Симптомы и последствия

- Отсутствие повторного использования в системах предприятия
- Отсутствие взаимодействия между системами предприятия
- Неспособность систем взаимодействовать даже когда используется одинаковые стандарты
- Чрезмерные расходы на техническое обслуживание из-за изменений бизнес-требований и необходимости внедрять новые продукты и технологии

# Дымоход предприятия (Stovepipe Enterprise)

- ◆ **Типичные причины**

- Отсутствие стратегии технологии предприятия
- Отсутствие стимулов для сотрудничества между разработчиками всех систем предприятия
- Отсутствие связи между проектами развития систем
- Недостаток знаний используемых стандартов и технологий
- Отсутствие горизонтальных интерфейсов в системе интеграционных решений

# Дымоход предприятия (Stovepipe Enterprise)

## ◆ Исключения

- Нельзя считать данным антипаттерном новые системы для только что организованных предприятий. Однако с течением времени, в результате укрупнения предприятия такой антипаттерн может иметь место
- Нельзя считать данным антипаттерном ситуацию, когда различные системы взаимодействуют через общую горизонтальную составляющую, например СУБД. Здесь возможно проявление антипаттерна Vendor Lock-In

# Дымоход предприятия (Stovepipe Enterprise)

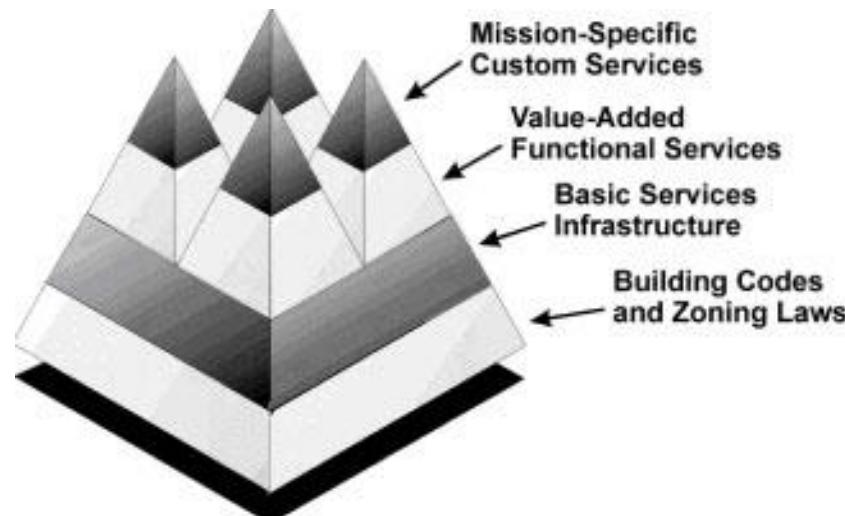
- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Дымоход предприятия (Stovepipe Enterprise)

## ◆ Как избежать?

- Координация технологий на нескольких уровнях за счет выбора стандартов взаимодействия
- Выбор стандартов может быть согласован через определения стандартов эталонной модели
- Создание общей операционной среды координирует выбор продуктов и контролирует конфигурацию версий продуктов



# Дымоход предприятия (Stovepipe Enterprise)

- ◆ **Как избежать?**
  - Весь опыт индустрии говорит о том, что большие системы уровня предприятия хорошо определяются объектно-ориентированной архитектурой.
  - Ключевой задачей крупномасштабной архитектуры является определение детального соглашения взаимодействия между системами
  - Используется четыре модели требований и четыре модели спецификации, которые позволяют корректно масштабировать и решать проблемы совместимости

# Дымоход предприятия (Stovepipe Enterprise)

- ◆ **Связанные антипаттерны:**

- Изобретение колеса – ориентирован на отсутствие зрелости конструкций и реализаций, что вызвано отсутствие связи между разрабатываемыми проектами



Это научный эксперимент

DEMOTIVATORS.RU

# Беспорядок



# Беспорядок (Jumble)

## ◆ Исток

- Вертикальные и горизонтальные элементы архитектурного дизайна перемешаны, что приводит к нестабильной и неясной архитектуре

## ◆ Причины

- Непродуманная архитектура
- Ускорение



# Беспорядок (Jumble)

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Беспорядок (Jumble)

- ◆ **Как избежать?**
  - Выделить горизонтальные и вертикальные элементы дизайна и разбить систему на несколько слоев

# Дымоход



# Stovepipe System (Дымоход)

## ◆ Исток

- Также как и в Stovepipe Enterprise отсутствует координация подсистем, но в рамках одной системы
- Реализация системы является хрупкой, т.к. существует множество неявных зависимостей от конфигурации системы, от установки и от состояния системы

# **Stovepipe System (Дымоход)**

- ◆ **Симптомы и последствия**

- Большой разрыв между архитектурной документацией и ее реализацией
- Архитекторы не знакомы с основными аспектами интеграции решения
- Проект превысил свой бюджет и график без видимых причин
- Изменения требований приводят к большим расходам
- Система соответствует требованиям на бумаге, но не соответствует ожиданиям пользователей
- Пользователи должны придумывать обходные пути, чтобы справиться с ограничениями системы

# **Stovepipe System (Дымоход)**

- ◆ **Симптомы и последствия**

- Несовместимость с другими системами и неспособность поддерживать интегрированность с другими системами
- Изменение в системе становятся все труднее
- Модификация системы все более вероятно приводит к новым серьезным ошибкам

# **Stovepipe System (Дымоход)**

- ◆ **Типичные причины**

- Множество инфраструктурных механизмов используется для интеграции подсистем, отсутствие общего механизма делает архитектуру более тяжелой для понимания и модификации
- Отсутствие абстракции, все интерфейсы уникальны для каждой подсистемы
- Недостаточное использование метаданных, метаданные не доступны для поддержки расширения системы без изменений в программном обеспечении
- Тесная связь между реализованными классами требует чрезмерного клиентского кода
- Отсутствие четкого понимания архитектуры системы

# **Stovepipe system (дымоход)**

- ◆ **Известные исключения**

- В целях исследования или прототипирования очень часто допускают данный антипаттерн, чтобы добиться быстрого решения

# **Stovepipe System (Дымоход)**

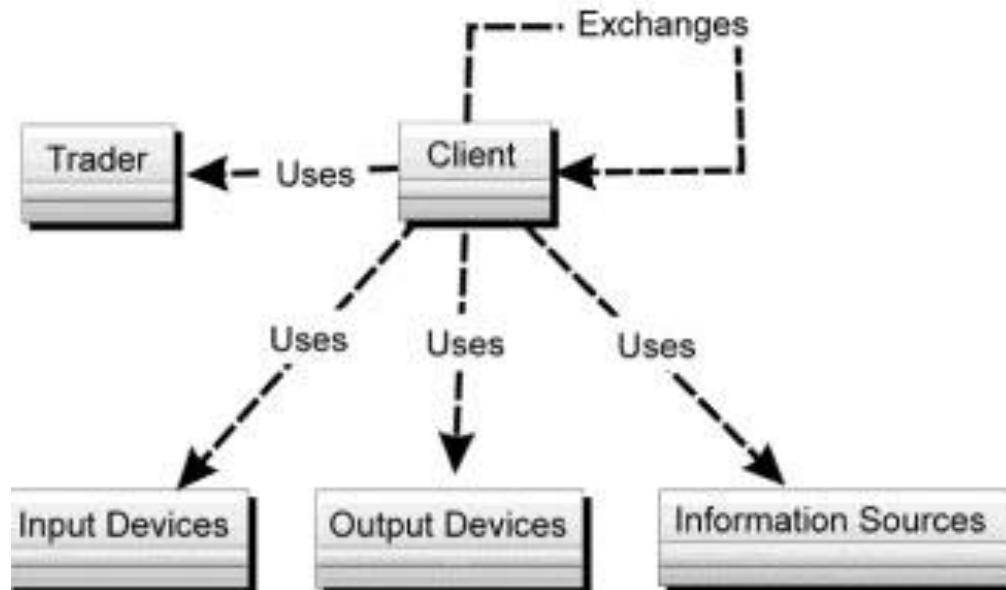
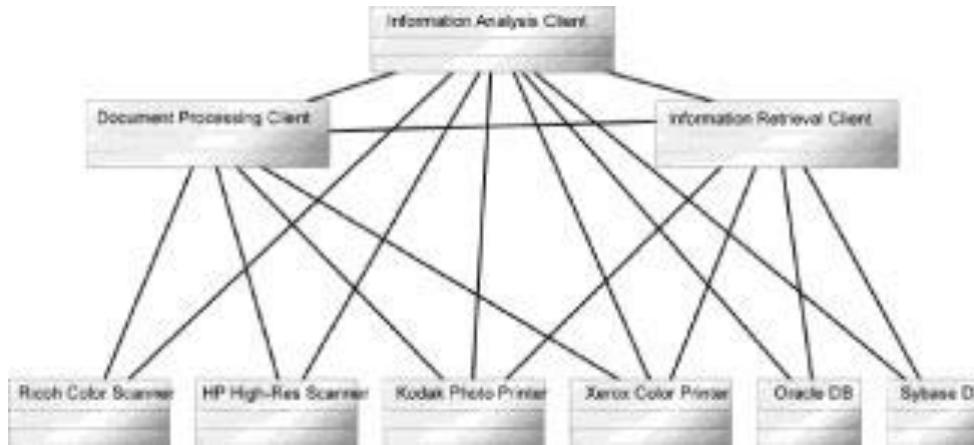
- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# **Stovepipe system (дымоход)**

- ◆ **Как избежать?**
  - Применить в рефакторинге компонентную архитектуру, которая обеспечивает гибкую подстановку программных модулей
  - Подсистемы моделируют абстрактно, используя для связи между ними интерфейсы.

# Stovepipe system (дымоход) – пример



# **Stovepipe system (Дымоход) – последствия**

- ◆ **Управленческие последствия**
  - Повышение рисков
  - Увеличение бюджета
- ◆ **Последствия для разработчиков**
  - Увеличение времени на проектирование и разработку системы
  - Постепенное увеличение сложности интерфейсов и проявление антипаттерна «Дымоход»

# Cover your assets



# Cover your assets

## ◆ Исток

- Стремясь избежать принятия важных решений с целью избегания ошибок, проектировщики берут безопасный курс, что приводит к абстрактному загадочному решению, которое не может передать намерения авторов

## ◆ Причины

- Нет ориентиров для принятия решений
- Приоритеты требований не расставлены

# Cover your assets

## ◆ Причины и последствия

- Когда никакие решения не принимаются и никаких приоритетов не установлено, то документы, описывающие требования имеют ограниченную ценность.
- Разработчики не знают, что и в какой последовательности следует реализовывать

# Cover your assets

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Cover your assets

- ◆ **Как избежать?**
  - Приоритезировать требования по согласованию с заказчиком
  - Разработать архитектурные модели, которые являются абстракциями информационных систем и способствуют связи между требованиями и техническим планом, между пользователями и разработчиками
  - Обеспечить, чтобы каждая система должна быть основана на архитектурной модели

# Vendor Lock-In



# Зависимость от поставщика (Vendor Lock-In )

- ◆ **Исток**
  - Архитектура корпоративных программ систем поставлена в зависимость от поставщика продукта

# Зависимость от поставщика (Vendor Lock-In )

## ◆ Симптомы и последствия

- Обновление коммерческих продуктов управляет циклом обслуживания программного обеспечения
- Обещанные характеристики продукта не реализуются/задерживаются поставщиками из-за чего нельзя обновить приложение
- Если пропущено обновление продукта, либо вышла его новая версия, то может потребоваться повторная покупка продукта и его реинтеграция с существующими системами

# Зависимость от поставщика (Vendor Lock-In )

## ◆ Типичные причины

- Продукт от поставщика не соответствует эталонной модели, потому что нет эффективного способа поиска соответствия между продуктом и эталонной моделью
- Продукт выбирается основываясь исключительно на маркетинговой информации без необходимого технического обследования
- Прикладное программирование требует углубленного знания продукта
- Сложность и общность технологии продукта значительно превышает потребности приложений
- Прямая зависимость от результатов, создаваемых продуктами, что не позволяет масштабировать применение этих продуктов

# Зависимость от поставщика (Vendor Lock-In )

- ◆ **Исключения**

- Ситуация, которая описана данным антипаттернам является приемлемой когда код одного поставщика используется всеми приложениями предприятия

# Зависимость от поставщика (Vendor Lock-In )

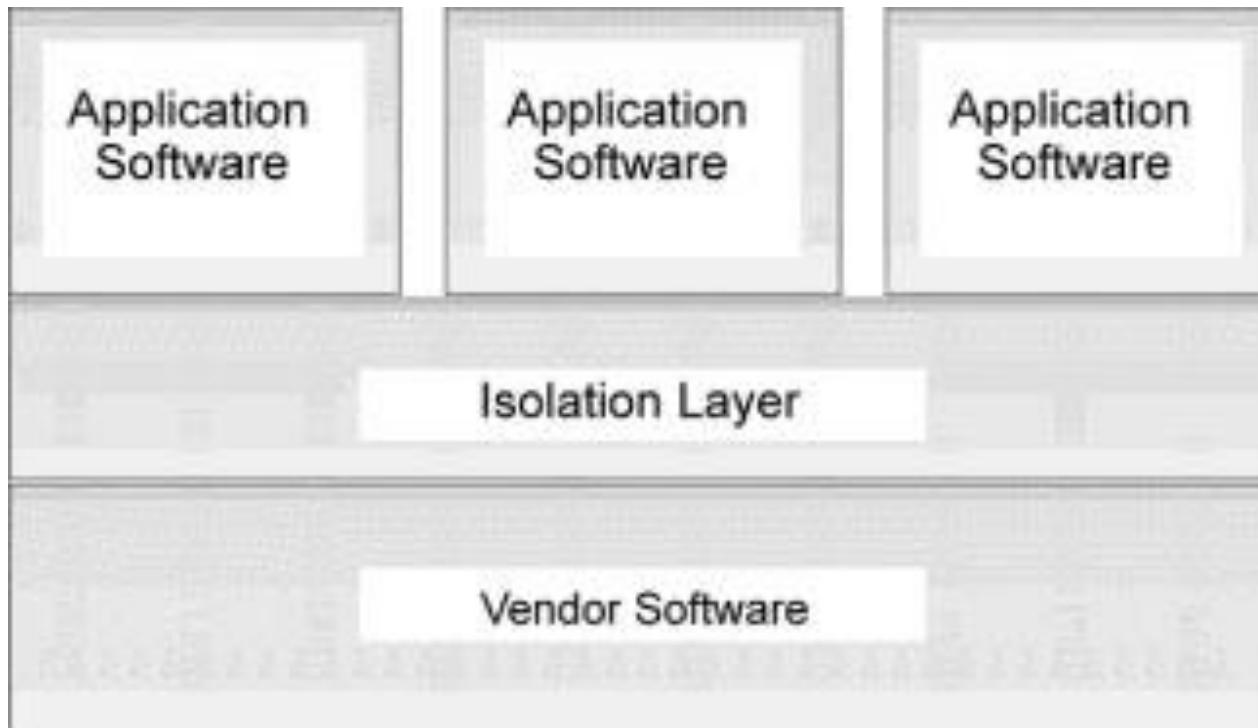
- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Зависимость от поставщика (Vendor Lock-In )

- ◆ **Как избежать?**

- Изолировать слой приложения от инфраструктуры низкого уровня (операционные системы, механизмы безопасности и т.д.)



# Зависимость от поставщика (Vendor Lock-In )

## ◆ Решение

- Изменения в лежащей в основе инфраструктуры планируются в течении всего жизненного цикла программного обеспечения, например планируется миграция перед выпуском новой версии продукта
- Необходимо использовать более удобный интерфейс прикладного программирования, для чего, возможно, потребуется создать промежуточный слой
- Поддержка нескольких инфраструктур в течении всего жизненного цикла программных продуктов

# Зависимость от поставщика (Vendor Lock-In )

## ◆ Влияние

- Может привести к потере контроля за развитием ИТ и диктату поставщика продукта
- Оказывает существенное влияние на управление рисками
- Знание для программирования внутри продукта могут устареть сразу после выхода новой версии продукта, что потребует обучения разработчиков и повысит вероятность их ошибок при написании кода



Это научный эксперимент

DEMOTIVATORS.RU

# Wolf Ticket



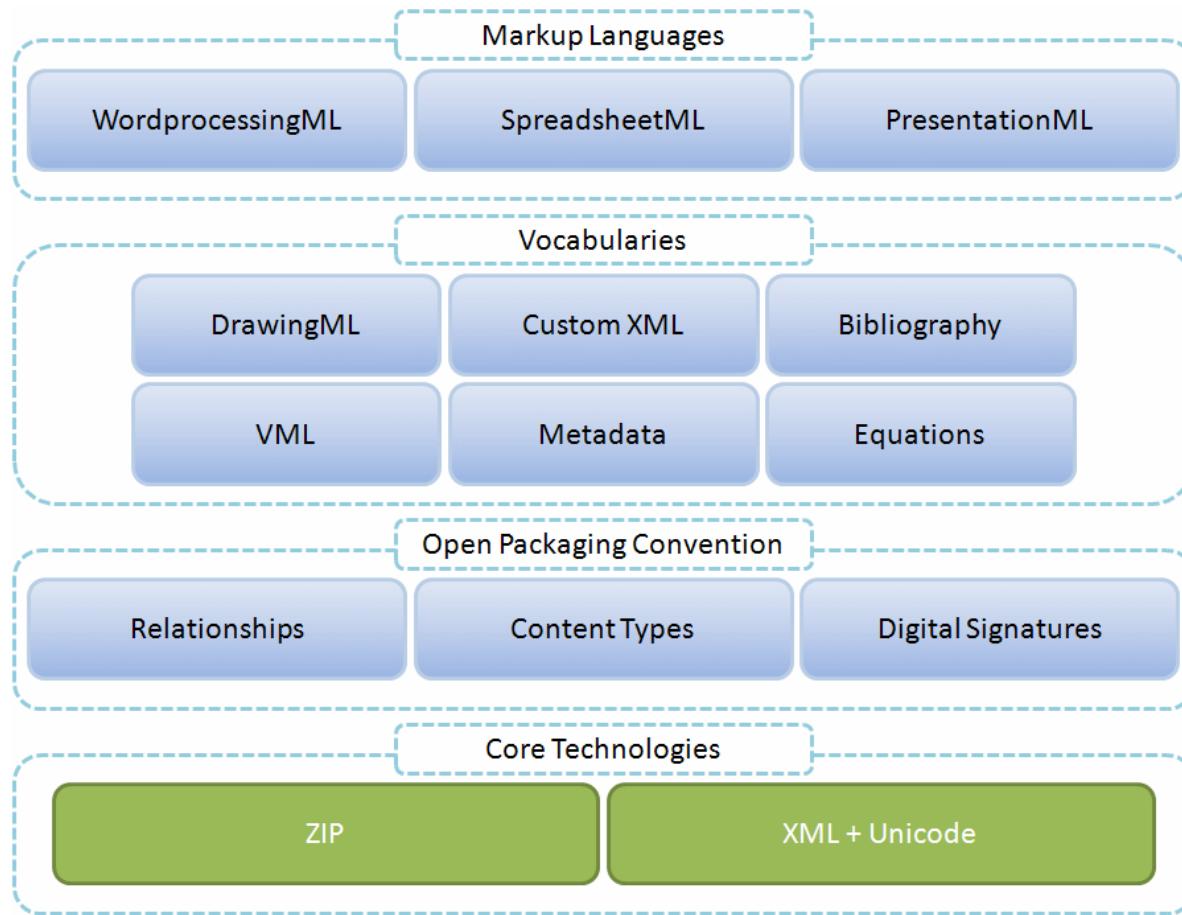
# Wolf Ticket

## ◆ Исток

- Стандартов на информационные системы гораздо больше, чем механизмов обеспечения соответствия программных систем этим стандартам
- Продукт поставляется с проприетарными интерфейсами, которые могут значительно отличаться от открытого стандарта.
- Открытые интерфейсы не всегда дают преимущества конечным пользователям

# Wolf Ticket. Пример //частное мнение 😊

- ◆ Формат Open XML Document в Microsoft Office



# Wolf Ticket

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Wolf Ticket

- ◆ **Как избежать?**
  - Для устранения технологических разрывов необходимо от поставщика требовать устранения разрыва между стандартами и реализацией до того, как продукт будет поставлен

# Architecture by Implication



# Архитектура по наитию (Architecture by Implication)

- ◆ **Исток**
  - Отсутствует архитектурная спецификации системы, которая уже находится на стадии реализации

# Архитектура по наитию (Architecture by Implication)

- ◆ Исток
  - Часто отсутствует определение архитектуры в следующих областях
    - ◆ Архитектура программного обеспечения, включающая используемые языки программирования, библиотеки, коды стандарта и т.д.
    - ◆ Аппаратная архитектура, включающая клиент и серверную конфигурацию
    - ◆ Коммуникационная архитектура, включающая сетевые протоколы и устройства
    - ◆ Архитектура хранения объектов, включающая СУБД и файлы
    - ◆ Архитектура безопасности
    - ◆ Архитектура управления системой

# Архитектура по наитию (Architecture by Implication)

## ◆ Симптомы и последствия

- Отсутствие архитектуры и технического задания: недостаточное определение архитектуры ПО, аппаратного обеспечения, коммуникаций, хранения данных и управление ПО
- Скрытые риски вызванные масштабированием, знаниями о предметной области, технологиями и любыми другими факторами
- Незнание новых технологий
- Отсутствие резервного копирования и планы действий в чрезвычайных ситуациях, например при отказе оборудования

# Архитектура по наитию (Architecture by Implication)

- ◆ **Типичные причины**

- Отсутствие управления рисками
- Самоуверенность менеджеров, архитекторов и/или разработчиков
- Опора на предыдущий опыт, который может отличаться в критических областях
- Неявные и не решенные вопросы в системной архитектуре, вызванные пробелами в системной инженерии

# Архитектура по наитию (Architecture by Implication)

- ◆ **Известные исключения**

- «Архитектура по наитию» допустима для повторяющихся решений, когда лишь небольшие различия в коде отличают одно решение от другого, например установочные скрипты.
- Подобное решение становится антипаттерном, когда его пытаются применить для новой предметной области, «протащив» существующие технологии в новую область без должного исследования возможностей их применения

# Архитектура по наитию (Architecture by Implication)

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Архитектура по наитию (Architecture by Implication)

## ◆ Решение

- Необходимо выработать подход к системной архитектуре и определить и ориентироваться на множество точек зрения на систему
- Каждая модель, отражает точку зрения определенного заинтересованного лица
- Точка зрения иллюстрируется таблицами, диаграммами или спецификациями, связанными в единое целое

# Архитектура по наитию (Architecture by Implication)

## ◆ Решение

- Шаги, необходимые для определения системной архитектуры используя определенную точку зрения:
  - ◆ **Определите цели архитектуры**
  - ◆ **Определить вопросы, которые могут быть адресованы заинтересованным лицам. Приоритезируйте запросы**
  - ◆ **Выберите точку зрения**
  - ◆ **Анализируйте каждую точку зрения**
  - ◆ **Трассируйте точку зрения на потребности заинтересованных лиц**

# Архитектура по наитию (Architecture by Implication)

- ◆ **Как избежать?**
  - Интегрируйте модели, чтобы убедиться что они не противоречат друг другу
  - Итерационно разрабатывайте архитектуру до тех пор, пока все вопросы и пробелы будут устранены

# Архитектура по наитию (Architecture by Implication)

## ◆ Как избежать?

- Предпринимайте активные усилия, чтобы связать в архитектуре ключевых заинтересованных лиц,
- создавайте документы, которые содержат ценную информацию на всех этапах разработки и поддержания жизненного цикла

# Архитектура по наитию (Architecture by Implication)

- ◆ **Как избежать?**
  - Проверяйте реализацию – определите любые связующие детали между моделью и реализацией системы
  - Примите решение о необходимости модификации системы или обновления модели.
  - Обеспечьте целостность документации



Это научный эксперимент

DEMOTIVATORS.RU

# Warn Bodies



# Warn Bodies

- ◆ **Исток**
  - Ошибочно привлекается дополнительный персонал при изменении требований программного обеспечения

# Warn Bodies

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Warn Bodies

## ◆ Как избежать?

- Идеальный размер проекта 4 программиста, идеальная продолжительность проекта 4 месяца
- В проектных командах, которые разрастаются более 5 человек возникает проблема координации и поддержания общего видения проекта
- В малых проектных группах с индивидуальными ответственностями гораздо больше шансов произвести успешное программное обеспечение

# Дизайн комитетом



# Дизайн комитетом

- ◆ **Исток**

- Группа участников объединяется для создания чего-либо при плохом или некомпетентном руководстве

# Дизайн комитетом

## ◆ Причины и последствия

- Излишняя сложность проекта
- Логические противоречия
- Отсутствие целостной структуры
- Задача, базовые спецификации и технические критерии отступают на второй план перед личными интересами участников
- Составные части продукта плохо взаимодействуют друг с другом

# Дизайн комитетом

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Дизайн комитетом

## ◆ Решение

- Изменить подход к совещаниям:
  - ◆ **участники сами должны уметь контролировать время без часов.**
  - ◆ **Комментарии должны быть не более 25 слов,**
  - ◆ **подробности только по запросу.**
  - ◆ **Должна быть четко сформулирована повестка дня**
  - ◆ **Необходимо дать ответ на вопросы «Зачем мы здесь?» и «каких результатов мы хотим?»**

# Дизайн комитетом

- ◆ **Решение**
  - Три типа совещания:
    - ◆ **Дивергентное – генерация идей для последующего использования**
    - ◆ **Конвергентные – принятие решений, часто путем консенсуса**
    - ◆ **Обмен информацией – обмен информацией между сотрудниками, включающий анализ, презентацию и/или обучение**
  - Четко распределить роли в команде: архитектор, разработчик, тестировщик. Менеджер проекта и т.л.
  - Каждый разработчик обычно отвечает за одну подсистему в общей системе и юнит-тестировании



Это научный эксперимент

DEMOTIVATORS.RU

# Swiss Army Knife (Interface bloat)



# Swiss Army Knife (Interface bloat)

- ◆ **Исток**

- Разработчик предусмотрел интерфейс класса, призванный охватить все варианты его использования. В результате создается интерфейс с большим количеством сигнатур в бесполезной попытке учесть все необходимое



# Swiss Army Knife (Interface bloat)

## ◆ Симптомы

- Изготовление интерфейса очень мощным и очень трудным для осуществления

## ◆ Причины

- Нарушение принципа разделения интерфейса
- Непродуманная архитектура
- Частая смена разработчиков



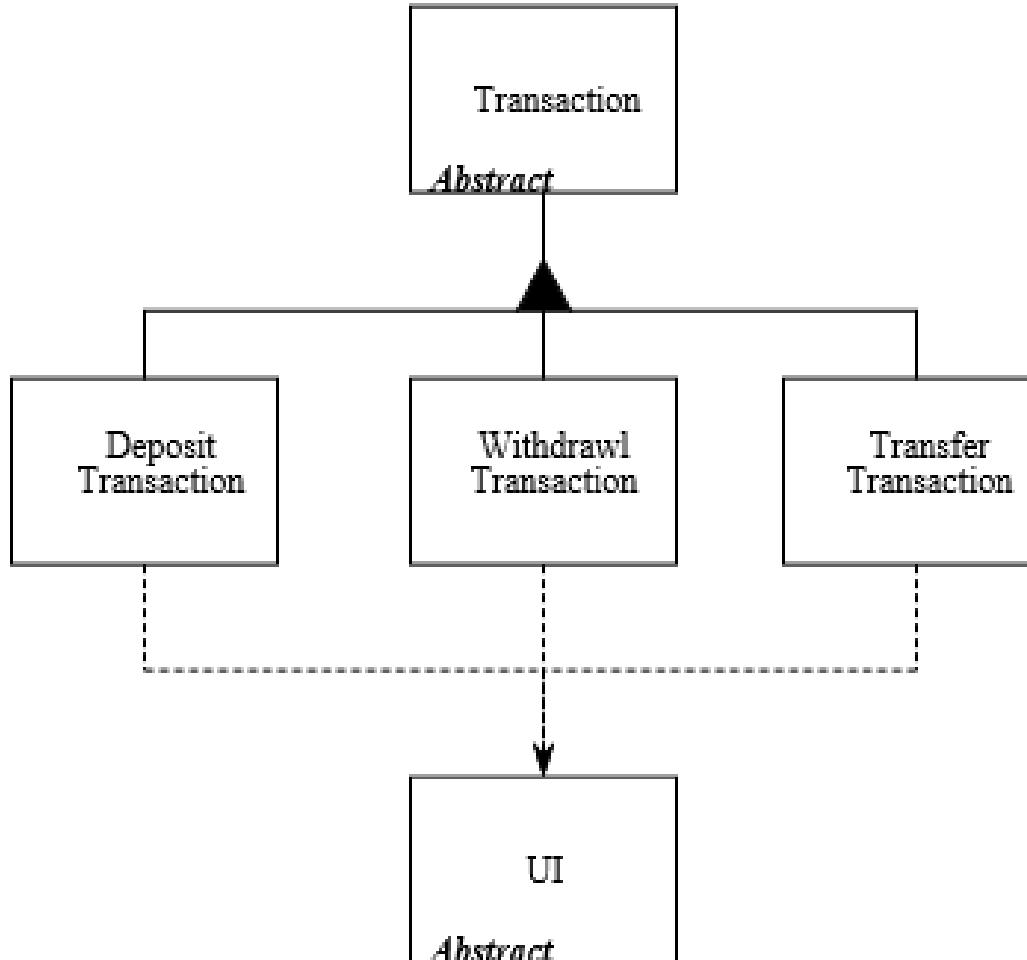
# Swiss Army Knife (Interface bloat)

- ◆ **Причины и последствия**

- Игнорируется возможность управления сложностью
- Интерфейс труден для понимания другими разработчиками
- Трудность отладки, документирования и технического обслуживания

# Swiss Army Knife (Interface bloat) // проблема

ATM Transaction Hierarchy



# Swiss Army Knife (Interface bloat)

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Важно! Принцип разделения интерфейса

- ◆ Клиент не должен вынуждено зависеть от интерфейса, который не использует

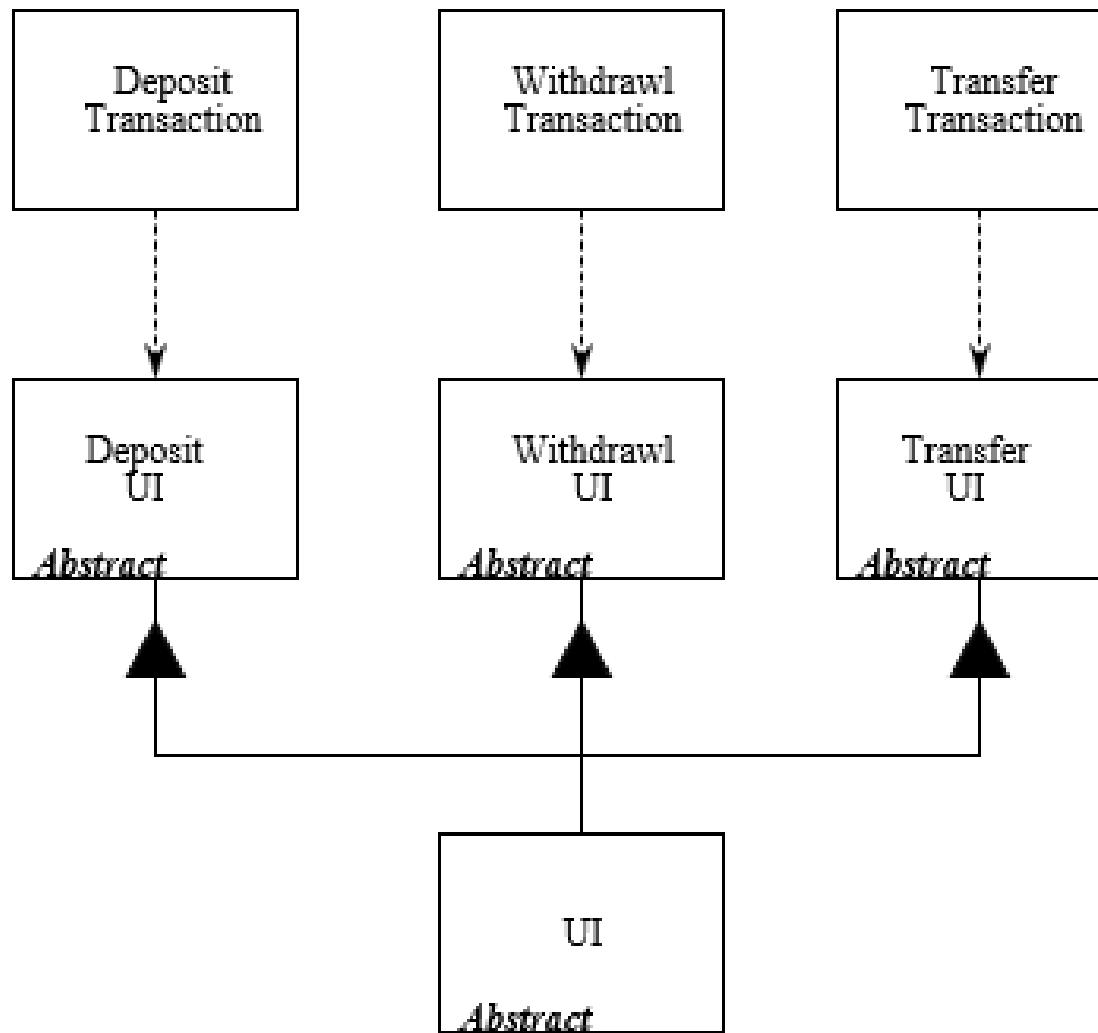
# Swiss Army Knife (Interface bloat)

## ◆ Решение

- Разделить объявление интерфейса и его реализацию таким образом, чтобы вы могли легко контролировать какие части вам действительно нужно
- Все зависимости от других интерфейсов должны быть автоматически решены поставщиком библиотеки, а не ее пользователем методом проб и ошибок
- При обновлении интерфейса учитывайте обратную совместимость путем создания новой версии интерфейса при любых изменениях существующего интерфейса

# Swiss Army Knife (Interface bloat) // Решение

Segregated ATM UI Interface



# Swiss Army Knife (Interface bloat)

- ◆ **Как избежать?**
  - Необходимо определить соглашение по разработке и использованию интерфейсов
    - ◆ Цель и задачи, решаемые каждым интерфейсом
    - ◆ Правила наименования интерфейса
    - ◆ Правила декларирования сигнатур интерфейса
    - ◆ Правила обработки исключений

# Swiss Army Knife (Interface bloat)

- ◆ **Взаимосвязь с другими антипаттернами**
  - Swiss Army Knife отличается от Blob тем, что может быть несколько классов, соответствующих данному антипаттерну, тогда как Blob это одиночный объект, инкапсулирующий как процесс, так и данные



Это научный эксперимент

DEMOTIVATORS.RU

# Reinvent The Wheel



# Reinvent The Wheel

- ◆ **Исток**

- Построение программной системы «с нуля», без повторного использования уже существующих решений, что приводит к разработке систем, впоследствии превращающихся в антипаттерн Stovepipe System

# Reinvent The Wheel

- ◆ **Причины и последствия**

- Закрытая системная архитектура
- Повторение функций коммерческого программного обеспечения
- Незрелые и неустойчивые требования и архитектурные решения

# Reinvent The Wheel

## ◆ Причины и последствия

- Отсутствие надлежащей поддержки управления изменениями и взаимодействия
- Длительные циклы развития проекта, прежде чем неудачные и тупиковые решения станут зрелыми и достаточными для долгосрочного развития системы и поддержки проекта
- Плохое управление рисками и затратами, ведущими к нарушению графика и перерасходованию средств
- Невозможность обеспечить желаемые характеристики для конечного пользователя, необходимость предпринимать значительные усилия чтобы повторить уже существующую функциональность в других системах

# Reinvent The Wheel

## ◆ Типичные причины

- Отсутствует коммуникация между разработчиками и передача знаний по технологиям
- Отсутствует явный процесс проектирования, который включает разработку модели предметной области и архитектурный анализ
- Предположение, что система будет построена с нуля
- Отсутствие общего взгляда на корпоративные системы, что ведет к появлению уникальных программных интерфейсов в каждой системе

# Reinvent The Wheel

- ◆ **Известные исключения**

- Решение, которое разрабатывается для исследования окружения, а также решение, в котором снижены затраты на коммуникацию между людьми с разными уровнями знаний, входящими в распределенную команду нельзя считать данным антипаттерном

# Reinvent The Wheel

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Reinvent The Wheel

- ◆ **Как избежать?**
  - Архитектурный анализ является способом быстрого успешного создания объектно-ориентированных решений, которые являются надежными, не зависят от продукта и повторно используемы
  - Вся информация извлекается из требований, которые представлены вариантами использования в объектно-ориентированной модели анализа
  - При анализе необходимо учитывать опыт уже существующих систем и проектных решений (паттерны)



Это научный эксперимент

DEMOTIVATORS.RU

# The Grand Old Duke of York



# The Grand Old Duke of York

## ◆ Исток

- Архитекторы и аналитики могут хорошо выделять абстрактные слои в программном обеспечении
- Не все разработчики могут оценить значимость хороших абстракций и оценят их значимость
- В результате разработчики строят свои абстракции, что делает проект, в итоге, сложным для понимания, расширения и технического обслуживания

# The Grand Old Duke of York

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# The Grand Old Duke of York

- ◆ **Как избежать?**
  - Сделать разделение ролей более эффективным
    - ◆ Аналитики и архитекторы – абстрагирование, разработка интерфейсов
    - ◆ Разработчики компонентов – создают инфраструктуру программного обеспечения и многоразовые компоненты
    - ◆ Разработчики приложений – интегрируют компоненты для создания рабочих систем. Могут использовать скриптовые языки высокого уровня (Visual Basic, Java Script, Perl и т.д.)

# Антипаттерны в разработке программного обеспечения

Модуль 3



# Антипаттерны в разработке ПО

- ◆ Хорошая структура кода программного обеспечения обеспечивает легкое понимание кода и расширение системы
- ◆ Рефакторинг программного обеспечения является эффективным подходом для улучшения структуры существующего кода
- ◆ Антипаттерны в разработке ПО используют формальные и неформальные подходы к рефакторингу кода

# Code smells и рефакторинг по Фаулеру

Classes	Within	Measured	Names	Unnecessary Complexity	Duplication	Conditional Logic
		Comments Long Method Large Class Long Parameter List	Type Embedded in Name (including Hungarian) Uncommunicative Name Inconsistent Names	Dead Code Speculative Generality	Magic Number Duplicated Code Alternative Classes with Different Interfaces	Null Check Complicated Boolean Expression Special Case Simulated Inheritance (Switch Statement)
Between	Data	Inheritance	Responsibility	Accommodating Change	Library Classes	
	Primitive Obsession Data Class Data Clump Temporary Field	Refused Bequest Inappropriate Intimacy (Subclass Form) Lazy Class Simulated Inheritance (Switch Statement) Parallel Inheritance Hierarchies Combinatorial Explosion	Feature Envy Inappropriate Intimacy (General Form) Message Chains Middle Man	Divergent Change Shotgun Surgery Parallel Inheritance Hierarchies Combinatorial Explosion	Incomplete Library Class	

## Database Smells

## Architecture Smells

Dependency Graphs	Inheritance Hierarchies	Packages	Subsystems	Layers
-------------------	-------------------------	----------	------------	--------

# Антипаттерны в разработке ПО

- ◆ **The Blob**
- ◆ **Continuous Obsolescence**
- ◆ **Lava Flow**
- ◆ **Ambiguous View Point**
- ◆ **Functional Decomposition**
- ◆ **Poltergeists**
- ◆ **Boat Anchor**
- ◆ **Golden Hammer**



# Антипаттерны в разработке ПО

- ◆ **Dead End**
- ◆ **Spaghetti Code**
- ◆ **Input Kludge**
- ◆ **Walking through a Minefield**
- ◆ **Copy-and-paste programming**
- ◆ **Error hiding/Exception handling**



# THE BLOB



# The Blob

## ◆ Исток

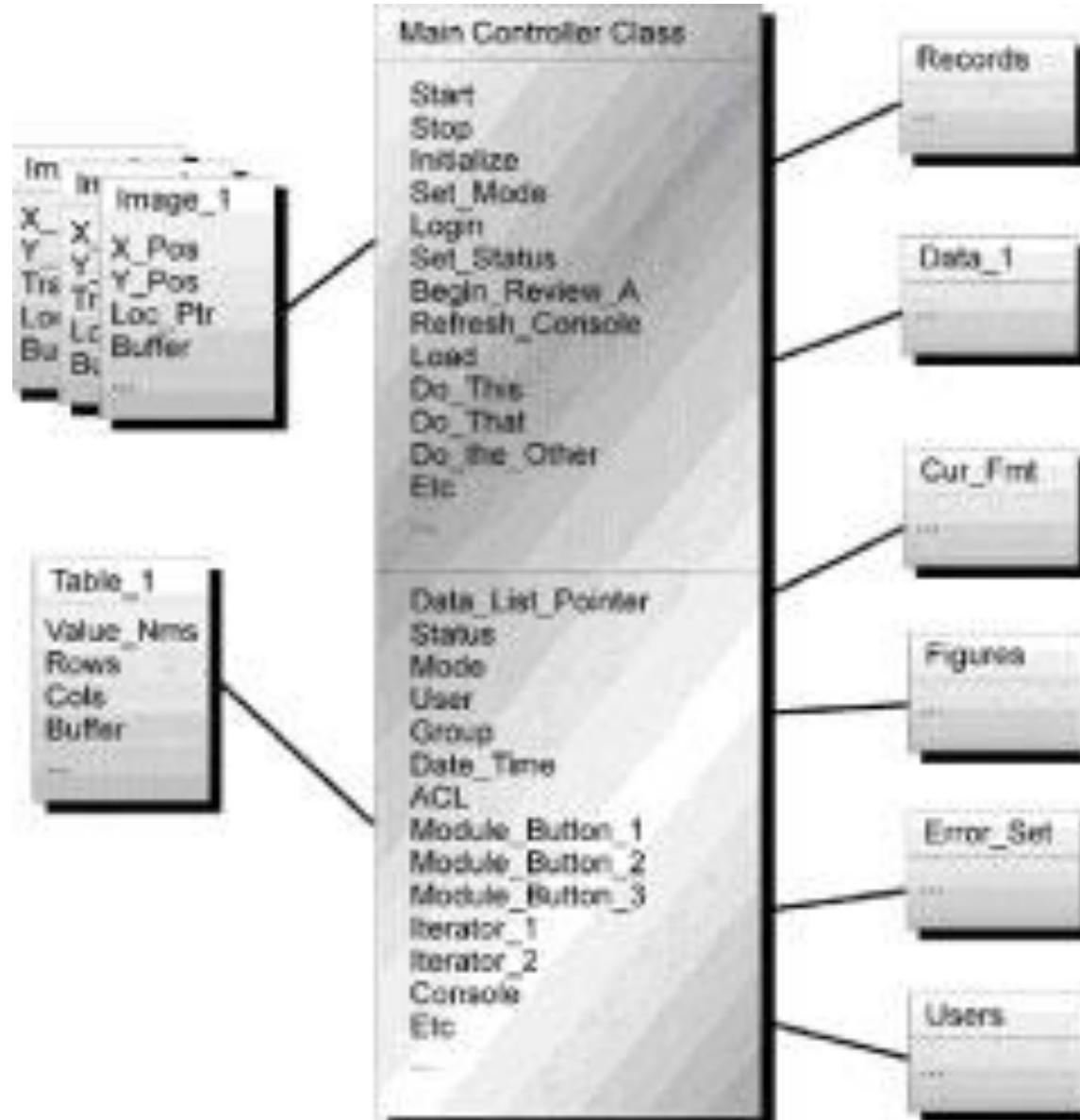
- Создан один единственный класс, который реализует различные ответственности и характеризуется наличием большого числа атрибутов и методов
- Проблема состоит в распределении ответственостей, которые взял на себя единственный класс

# The Blob

## ◆ Симптомы и последствия

- Один класс с большим количеством атрибутов и операций.  
Класс с 60 и более атрибутами и операциями
- Несоизмеримая коллекция не связанных атрибутов и операций, инкапсулированных в одном классе, отсутствие взаимосвязи между атрибутами и операциями
- Отсутствие объектно-ориентированного дизайна. Вся логика программы сосредоточена в одном методе, который связан с пассивными объектами данных.
- Унаследованный процедурный дизайн, который не был изменен в соответствии с объектно-ориентированной архитектурой

# The Blob



# The Blob

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# The Blob

## ◆ Решение

- Примените рефакторинг «Извлечение класса», «Извлечение метода», «Выделение подкласса/интерфейса»
- Предпочитайте минимальные классы монолитным



Это научный эксперимент

DEMOTIVATORS.RU

# Continuous Obsolescence



# Continuous Obsolescence

## ◆ Исток

- Развитие программных технологий ведет к непрерывному устареванию кода и необходимости поиска способов поддерживать устаревшие и новые технологии в рамках одного продукта

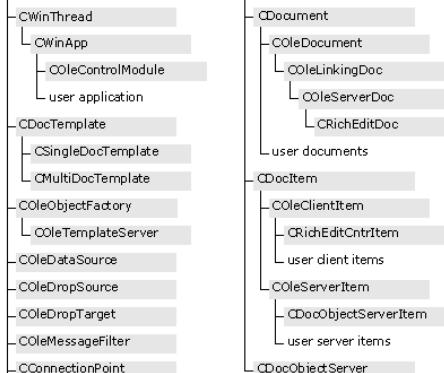
# Continuous Obsolescence

Microsoft Foundation Class Library Version 4.21

## CObject

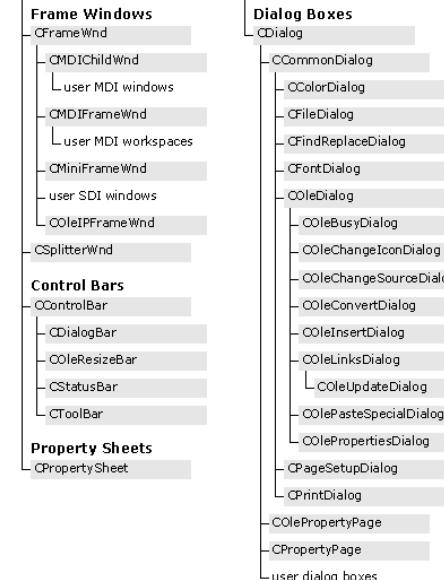
### Application Architecture

#### CComTarget

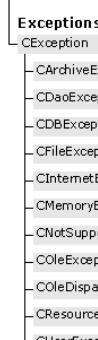


### Window Support

#### CWnd

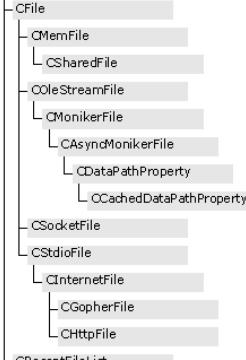


### user objects



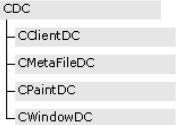
### Exceptions

### File Services



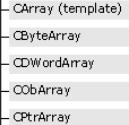
### File Services

### Graphical Drawing



### Graphical Drawing

### Arrays

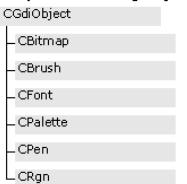


### Arrays

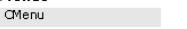
### Control Support



### Graphical Drawing Objects



### Menus



### ODBC Database Support



### DAO Database Support



### Maps



### Maps

### OLE Type Wrappers



### Structures

### OLE Automation Types



### Synchronization



### Classes Not Derived from CObject

#### Internet Server API



#### Run-time Object Model Support



#### Simple Value Types



#### Typed Template Collections



#### OLE Type Wrappers



#### Structures



#### OLE Automation Types



#### Synchronization



# Continuous Obsolescence

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Continuous Obsolescence

- ◆ **Решение**

- Применить рефакторинг «Выделение интерфейса», чтобы ослабить зависимость от технологии, которая может быстрее всего измениться



Это научный эксперимент

DEMOTIVATORS.RU

# Lava Flow



# Lava Flow

## ◆ Исток

- Программная система разработана несколькими людьми, уже давно не работающими в организации, код запутанный, трудноподдерживаемый
- Для экономии времени новые разработчики не пытаются исправить код, а дописывают новый код, который только усугубляет ситуацию

# Lava Flow

Генеральный (*в отдел кадров*). Уволить этих бездарей! Найдите мне нормальных специалистов!

...спустя три недели новые специалисты сидят на рабочих местах и изучают проект.

Junior developer (*про себя*). Как тут все сложно... Жизни не хватит чтобы понять как все это работает... Меня наверно уволят...

Senior developer (*громко и недовольно*). Кто учил этих мудаков программировать?!! Нет, ну вы посмотрите что они тут пишут!

Главный архитектор (*задумчиво*). Если бы это была лошадь, я бы посоветовал ее пристрелить...

Team leader (*нервно*). Спокойно, мужики. Заказчик хочет чтобы мы просто кое-что тут дописали. Оставим все как есть, просто доделаем то, что требуется.

Project manager (*обращаясь к разработчикам*). Проект конечно тяжелый, но мы ведь справимся! Тем более мы будем вести разработку по самой лучшей методологии! Нам поможет XP! Scrum! TDD! (нужное подчеркнуть).

Главный архитектор (*поймав project manager-а в курилке*). Ты ведь понимаешь, что проект уже бьется в предсмертных конвульсиях. Его нужно полностью переписать.

Project manager (*Главному архитектору*). Да все я понимаю, только кто мне это позволит? Знаешь сколько уже потрачено бабла на этот проект? Попробуй объяснить генеральному, что нужно все переделать, я посмотрю насколько быстро ты отыщешь живописное местечко под названием «нах#й».

Project manager (*Оставвшись один в курилке, глядя в окно*). Вот начнем мы писать проект заново. И облажаемся. Я же во всем виноват буду, с меня же шкуру спустят. Ну его нафиг. А так, если сроки и завалим, то я всегда смогу свалить все на то, что проект уже был полным говном, когда достался нам.

...спустя полгода.

Генеральный (*в отдел кадров*). Уволить этих бездарей! Найдите мне нормальных специалистов!

# Lava Flow

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Lava Flow

## ◆ Как предотвратить?

- Команда всегда должна понимать, как работает программный продукт. Нельзя работать с непонятным кодом
- Пишите тесты. Тесты позволяют понять код и найти скрытые ошибки
- Выполняйте рефакторинг кода, который достался вам в наследство

# Ambiguous View Point



$$\Psi_{\text{kitty}} = \frac{1}{\sqrt{2}} \Psi_{\text{alive}} + \frac{1}{\sqrt{2}} \Psi_{\text{dead}}$$

# Ambiguous View Point

ЗАМОК

КАЗНИТЬ

НЕЛЬЗЯ

ПОМИЛОВАТЬ

# Ambiguous View Point

- ◆ **Симптомы**

- Представление модели без спецификации её точки рассмотрения

- ◆ **Причины**

- Разработчик разработал модель без четкого понимания цели ее создания

# Ambiguous View Point

- ◆ Сталкивались ли вы с анти-паттерном «Ambiguous View Point»?
- ◆ Как вы решили эту проблему?



# Ambiguous View Point

## ◆ Решение

- Сформулировать цель построения модели
- Сформулировать вопросы, на которые должна дать ответ модель
- Полностью переделать модель

# Ambiguous View Point

## ◆ Исток

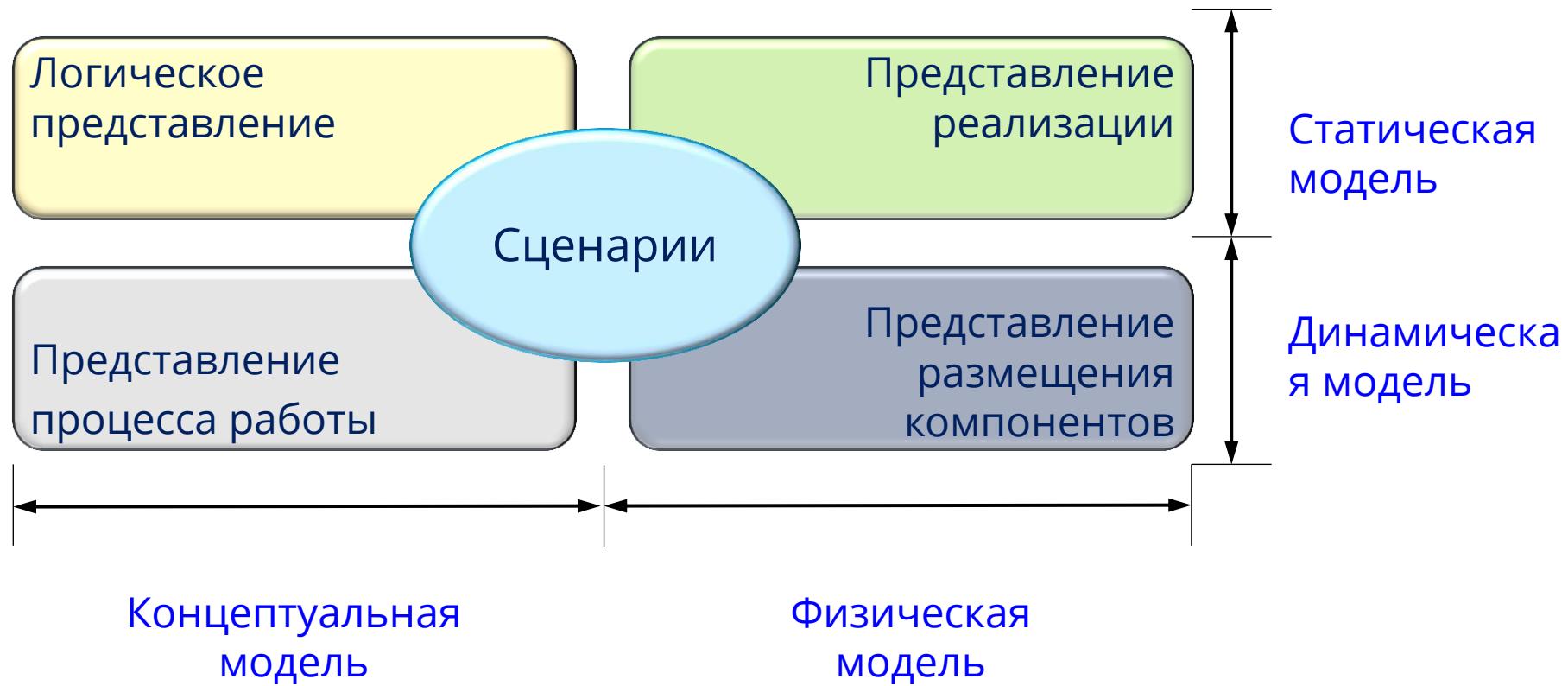
- Объектно-ориентированный дизайн построен без учета точки зрения на модель (абстракция/реализация и т.д.)
- Смешение точек зрения теряет преимущество объектно-ориентированного подхода

# Ambiguous View Point

- ◆ Сталкивались ли вы с анти-паттерном «Неопределенная точка зрения»?
- ◆ Как вы решили эту проблему?



# Ambiguous View Point



# Ambiguous View Point

- ◆ **Решение и как предотвратить?**

- Определить цель построения модели
- Определить точку зрения модели
- Проверить модель на соответствие цели и точки зрения
- Внести изменения в модель или создать новую

# Functional Decomposition



# Functional Decomposition

- ◆ **Исток**

- Приложение реализовано в процедурном стиле на объектно-ориентированном языке программирования

# Functional Decomposition

- ◆ **Симптомы и последствия**

- Классы содержат функциональные имена, например «Display\_Table»
- Все атрибуты класса приватные и используются только внутри класса

# Functional Decomposition

## ◆ Симптомы и последствия

- Абсолютно не принимаются во внимание такие принципы ООП как наследование и полиморфизм
- Нет документации из которой можно понять как система работает
- Нельзя повторно использовать код
- Пессимизм тестеров

# Functional Decomposition

- ◆ **Типичные причины**

- Отсутствие понимания у разработчиков объектно-ориентированного подхода
- Отсутствие архитектуры системы
- При постановке задачи приоритетной была архитектура, а не анализ требований

# Functional Decomposition

- ◆ **Известные исключения**

- Не все задачи требуют использования объектно-ориентированного подхода
- Такие задачи, например математические расчеты не являются проявлением антипаттерна Functional Decomposition

# Functional Decomposition

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?

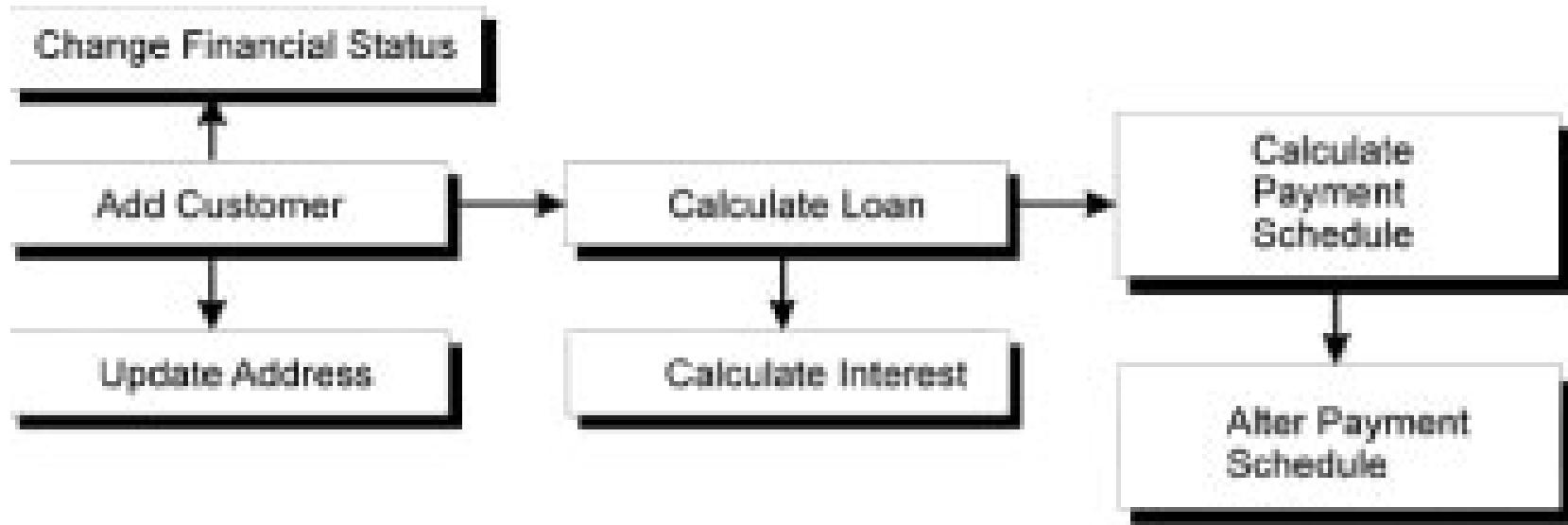


# Functional Decomposition

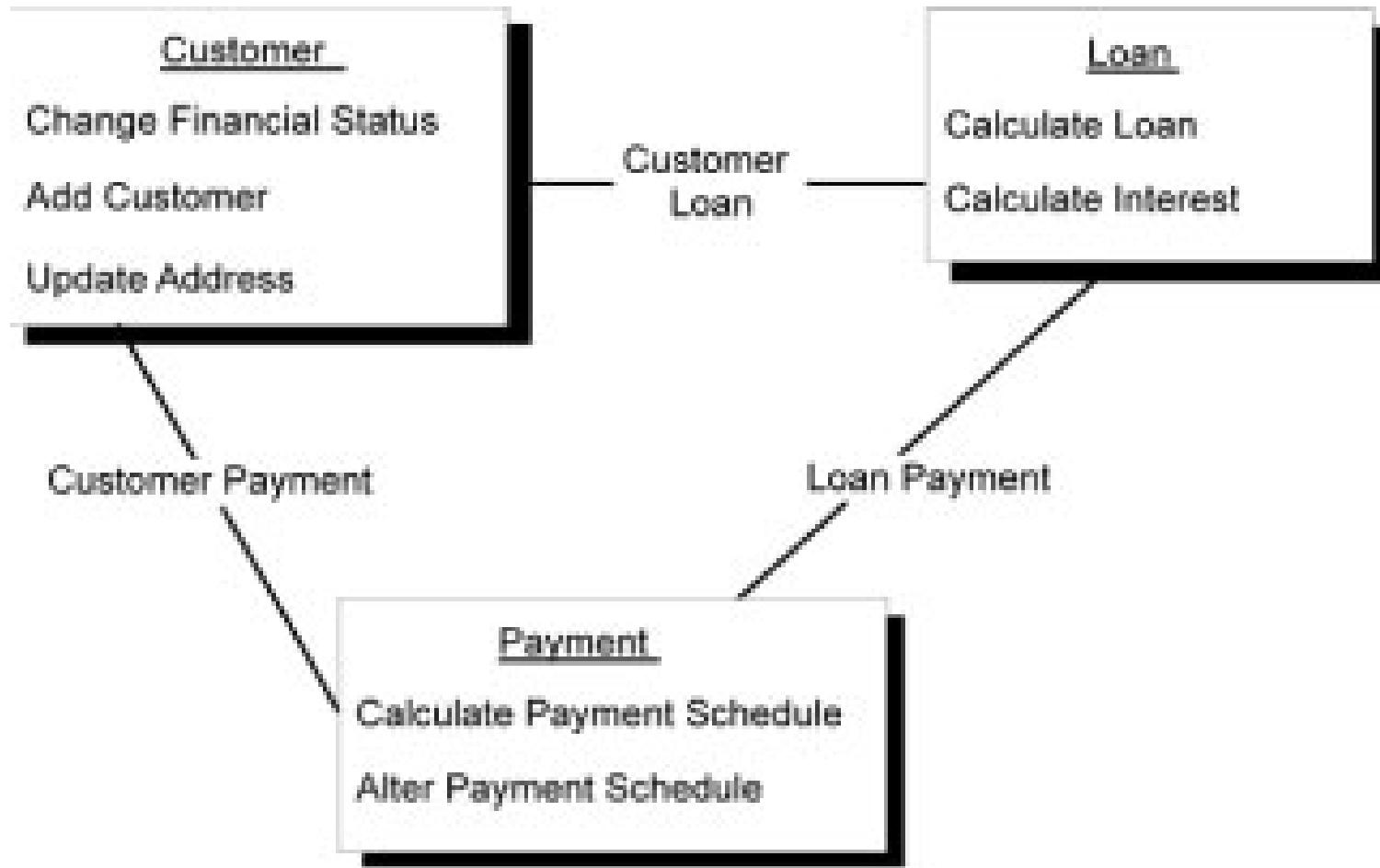
- ◆ **Решение и как предотвратить?**

- Получить модель системы с точки зрения пользователя (Use Case View Point)
- Выполните построение объектно-ориентированной модели, в которой учтите уже реализованные элементы системы, которые станут операциями объектов
- При необходимости выполните рефакторинг «Выделение метода», «Выделение класса», «Выделение интерфейса»

# Functional Decomposition



# Functional Decomposition



# Functional Decomposition

- ◆ Связь с другими антипаттернами
  - The Blob

# Poltergeists



# Poltergeist

## ◆ Исток

- Класс, у которого ограниченная ответственность и роль, которую он играет в системе
- создан не на основе абстракции предметной области из-за плохого понимания сути объектно-ориентированного подхода
  - ◆ **У таких классов небольшой жизненный цикл**
  - ◆ **Они не эффективны, т.к. используют избыточные связи**
  - ◆ **Они не связаны с абстракциями предметной области и захламляют объектную модель**

# Poltergeist

## ◆ Симптомы и последствия

- Избыточные связи с другими объектами
- Временные ассоциации
- Классы без состояний
- Классы, живущие короткое время, только для того, чтобы вызвать другие классы
- Классы с «процессными» именами, например «start\_process\_alpha»

# Poltergeist

- ◆ **Типичные причины**
  - Отсутствие объектно-ориентированной архитектуры
  - Неправильный выбор пути решения задачи
  - Предположения об архитектуре приложения на этапе анализа требований до объектно-ориентированного анализа
  - Страх добавления классов

# Poltergeist

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Poltergeist

```
import java.util.EmptyStackException;
import java.util.LinkedList;

public class LabStack<T> {
    private LinkedList<T> list;

    public LabStack() {
        list = new LinkedList<T>();
    }

    public boolean empty() {
        return list.isEmpty();
    }

    public T peek() throws EmptyStackException {
        if (list.isEmpty()) {
            throw new EmptyStackException();
        }
        return list.peek();
    }

    public T pop() throws EmptyStackException {
        if (list.isEmpty()) {
            throw new EmptyStackException();
        }
        return list.pop();
    }

    public void push(T element) {
        list.push(element);
    }

    public int size() {
        return list.size();
    }

    public void makeEmpty() {
        list.clear();
    }

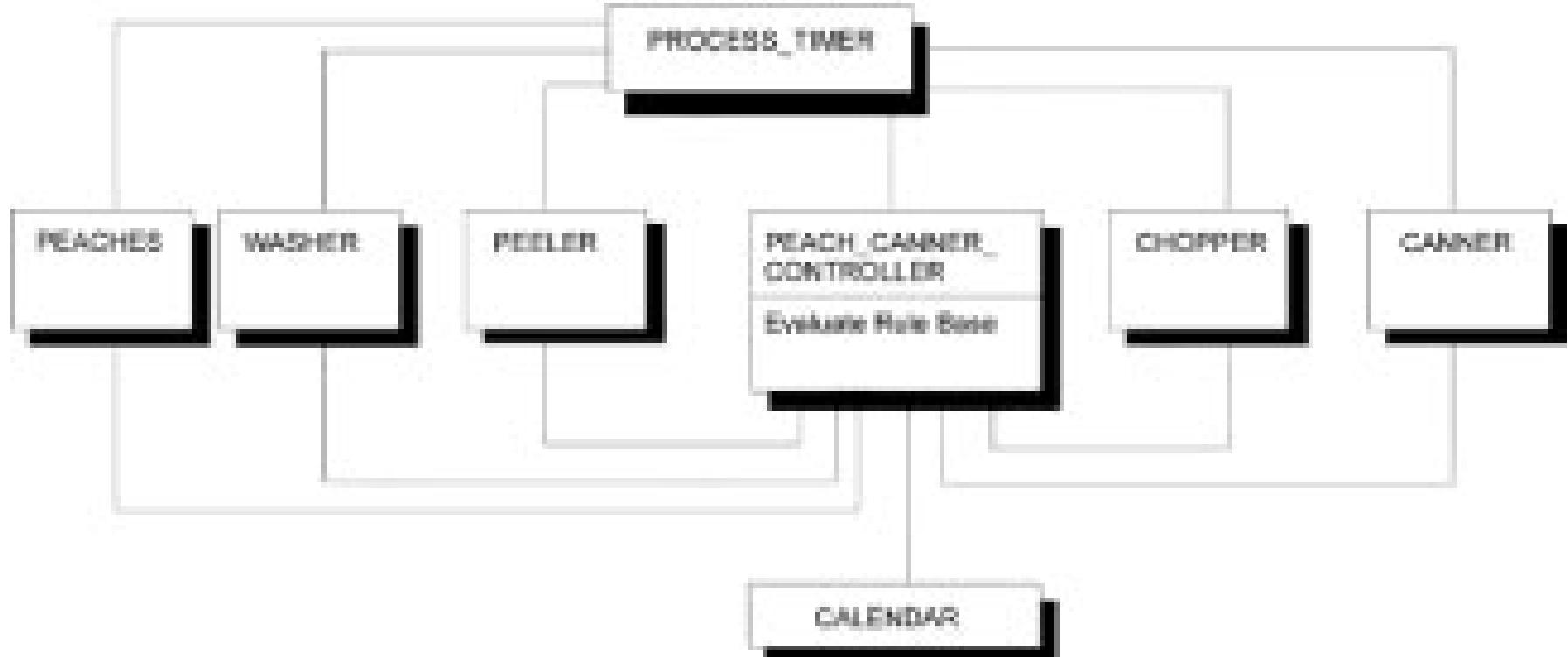
    public String toString() {
        return list.toString();
    }
}
```

# Poltergeist

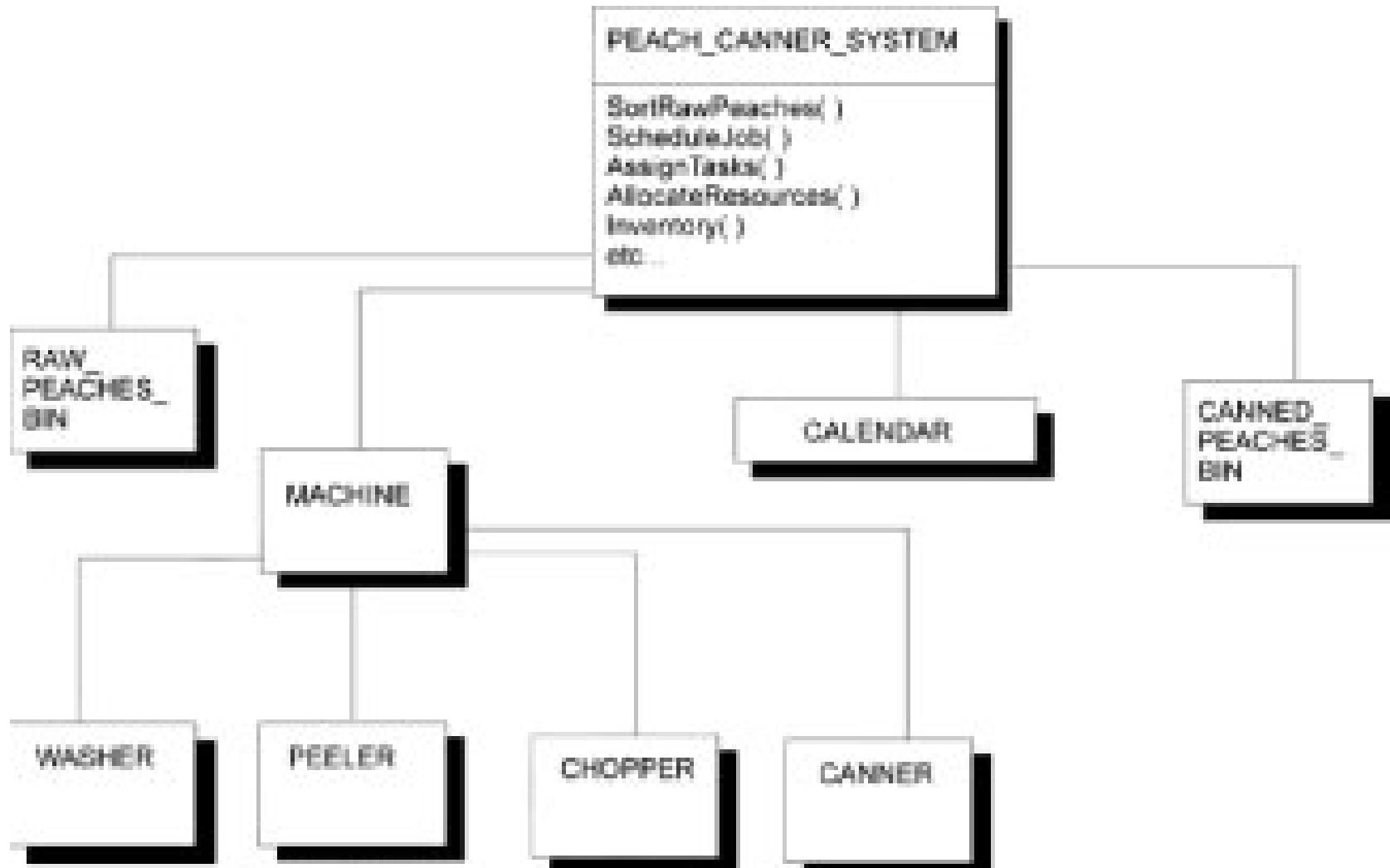
## ◆ Решение

- Удаление внешних классов (не имеющих отношения к системе)
- Удаление классов не относящихся к доменной модели
- Рефакторинг для устранения транзитивных классов
- Рефакторинг для устранения классов, содержащих только операции
- Рефакторинг других классов с коротким жизненным циклом или несколькими ответственностями
- Перенос управляющих действий полтергейста в классы, методы который он вызывает

# Poltergeist



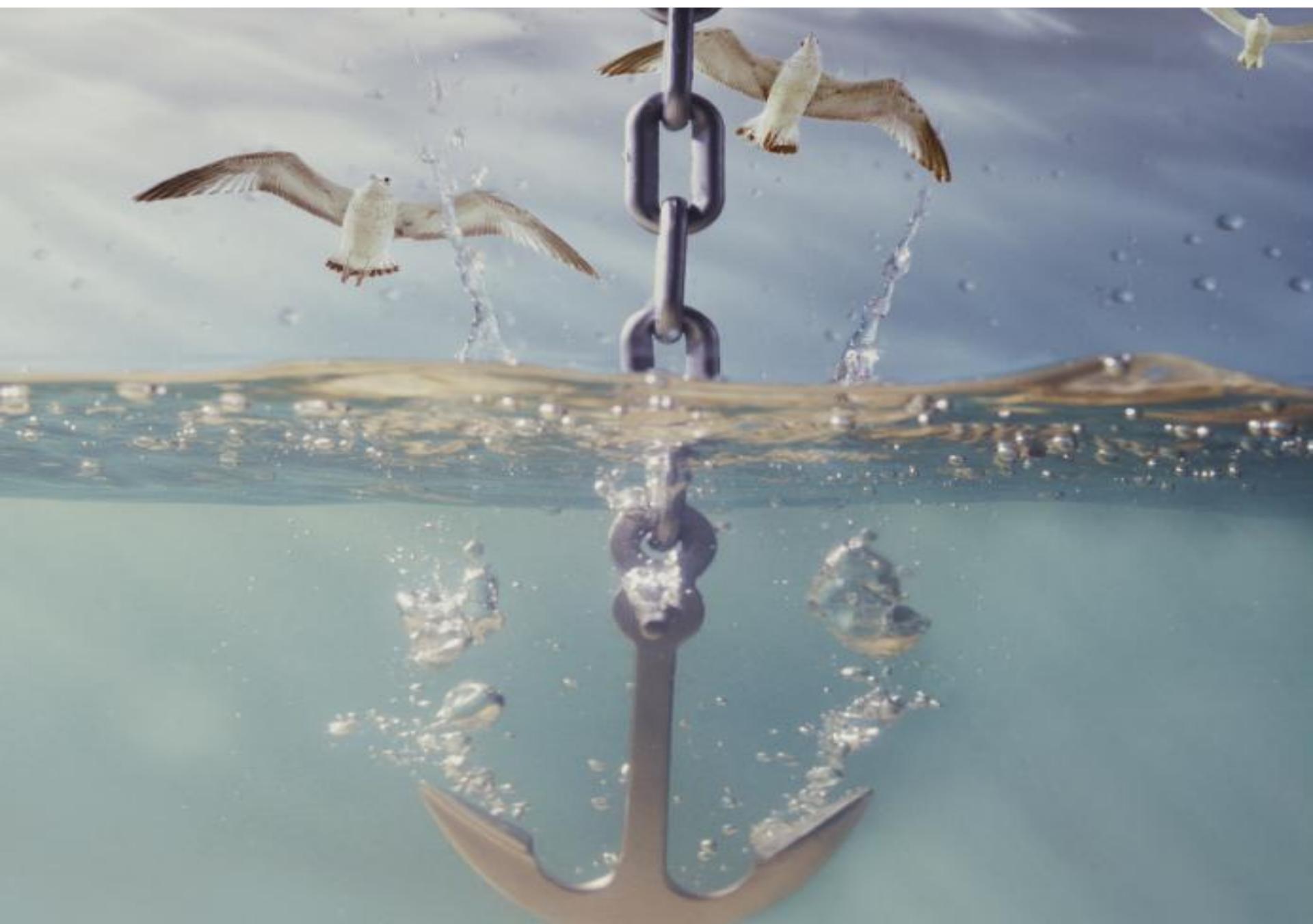
# Poltergeist



# Poltergeist

- ◆ Связь с другими антипаттернами
  - The Blob
- ◆ Как предотвратить?
  - Менеджеры должны заботиться о том, чтобы объектно-ориентированные архитектуры оценивались квалифицированными архитекторами на самом раннем этапе их создания и регулярно в последующем

# Boat Anchor



# Boat Anchor

## ◆ Исток

- Аппаратное или программное обеспечение, которое присутствует в системе не требуется для данного проекта
- В системе присутствует код, который не используется.

# Boat Anchor

## ◆ Причины

- Менеджеры или другие влиятельные лица убеждены в полезности данного аппаратного и/или программного решения
- Выводы по аппаратному или программному обеспечению сделаны без надлежащей оценки
- Программист не желает его удалять, мотивируя тем, что «когда-нибудь он пригодится»

# Boat Anchor

## ◆ Последствия

- Деньги на покупку потрачены впустую
- Потраченное впустую время на приобретение оборудования
- Может стать сдерживающим фактором в развитии
- Может вызвать проблемы при попытке заставить работать, что в итоге потребует отказа от этого решения, а значит время и усилия затраченные на интеграцию будут потрачены впустую
- Приводит к замусориванию кода, что делает его труднопонимаемым и затрудняет его поддержку

# Boat Anchor

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Boat Anchor

- ◆ **Как предотвратить?**
  - Аппаратное обеспечение
    - ◆ **Выполнить надлежащую оценку аппаратного обеспечения с точки зрения полезности в проекте, прежде чем покупать его**
    - ◆ **Иметь хотя бы грубый план использования программного обеспечения, прежде чем покупать его**
    - ◆ **Быть уверенным, что аппаратное обеспечение будет использовано в проекте с момента идеи до реализации**

# Boat Anchor

- ◆ **Как предотвратить?**
  - Программное обеспечение
    - ◆ **Выполните надлежащую оценку перспектив использования программного обеспечения, прежде чем покупать его**
    - ◆ **Удалите вызовы устаревшего/ненужного кода**
    - ◆ **Удалите неиспользуемый код**
    - ◆ **Удалите неиспользуемые библиотеки**
    - ◆ **Использовать системы управления версиями (Subversion, Git и т.д.)**



*Golden*  
**HAMMER**

# Golden Hammer

## ◆ Исток

- Применение какого-то конкретного паттерна для решения всех возможных и невозможных задач
- Рассматривается одно решение как универсальное на все случаи жизни

# Golden Hammer

## ◆ Симптомы и последствия

- Идентичные инструменты и продукты используются для широкого спектра задач
- Решения уступают по производительности, масштабируемости и т.д., в сравнении с другими аналогичными решениями
- Разработчики часто вступают в спор с системными аналитиками относительно требований к системе и пользователи часто их защищают
- Разработчики изолированы от реальных проектов, демонстрируют недостаток знаний и опыта в альтернативных подходах

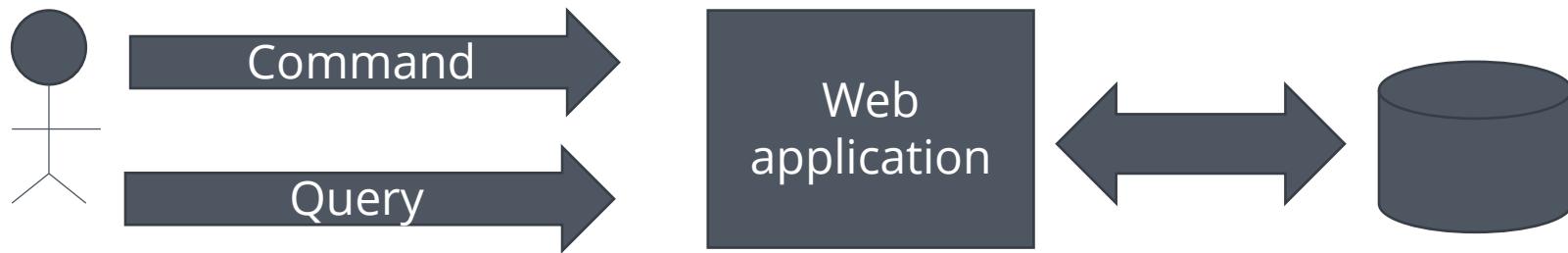
# Golden Hammer

## ◆ Причины и последствия

- «Влюбленность» в тот или иной паттерн
- Не желание у новичков изучать что-то новое, попытка использовать знакомые методы для решения других задач
- Ранее были сделаны большие инвестиции в обучении постоянно используемой в решениях технологии

# Golden Hammer

- ◆ Пример



# Golden Hammer

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Golden Hammer

- ◆ **Как избежать?**
  - Организация должна развивать стремление к новым технологиям путем профессионального развития своих разработчиков
  - Программное обеспечение должно быть спроектировано с учетом широкого применения интерфейсов между компонентами

# Golden Hammer

- ◆ **Связанные антипаттерны**

- Lava Flow
- Vendor Lock-In

# Dead End



# Dead End

## ◆ Исток

- Когда повторно используемые компоненты не поддерживаются более поставщиком бремя поддержки возлагается на разработчиков прикладного ПО
- После внесения изменений компонент не может быть легко интегрирован в систему
- Если же поставщик компонентов выпустит новую версию, то будет необходимо объединить изменения сделанные разработчиками прикладного ПО с изменения в компоненте.  
Это не всегда возможно!

# Dead End

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Dead End

- ◆ **Как избежать?**
  - Откажитесь от модификации повторно используемых компонент, если их поставщик когда-либо планирует выпустить обновление
  - Если модификация необходимо, создайте изолирующий уровень, чтобы ослабить зависимость от изменений (паттерн «Wrapper» или наследование через композицию)

# Dead End

- ◆ **Исключение**

- Dead End приемлем, если модификация выполняется с исследовательской точки зрения в тестовом коде с целью проверки каких-либо технических предположений

# Spaghetti code



# Spaghetti code

- ◆ **Симптомы**

- Система без четко выраженной структуры

- ◆ **Причины**

- Изменения требований
  - Смена разработчиков
  - Некомпетентность



# Spaghetti code

## ◆ Обнаружение

- Где реализуется определенная часть функционала?
- Где создается экземпляр объекта и как это происходит?
- Как определить критическую секцию для исправления?

# Spaghetti code

## ◆ СИМПТОМЫ

- Много методов класса без параметров
- Подозрительные или глобальные переменные
- Переплетные или непредвиденные связи между объектами
- Процессно-ориентированные методы, объекты с процессно-ориентированными именами
- Потеря преимуществ объектно-ориентированного подхода – полиморфизм не используется в полной мере

# Spaghetti Code

## ◆ Исключение

- Спагетти-код может быть приемлимым для случая, когда такой участок кода локализован в интерфейсе.
- Подобные компоненты имеют короткое время жизни и изолированы от остальной части

# Spaghetti code

## ◆ Как избежать?

- Получайте доступ к членам класса через функции доступа (properties)
- Преобразуйте фрагмент кода функции, которая может быть использована повторно. Не используйте «Копировать-и-вставить»!
- Измените порядок аргументов всех функций, чтобы достичь согласованности по всей кодовой базе
- Удалите участки кода, которые уже стали не используемыми, в противном случае вы получаете проявление паттерна Lava Flow
- Переименуйте методы, классы, переменные в соответствии со стандартом вашей компании/принятым подходом и отражающим вашу предметную область

# Spaghetti code

- ◆ **Как избежать? (полный реинжиниринг кода!)**
  - Настаивайте на качественном объектно-ориентированном анализе предметной области. Доменная модель является основой
  - После объектно-ориентированного анализа из системных требований станет ясна степень изменчивости

# Spaghetti code

- ◆ **Рефакторинг**
  - Создать абстрактный суперкласс
  - Упросить условия
  - Реорганизация классов за счет композици

# Input Kludge

A dark blue background featuring a faint, diagonal watermark of binary code (0s and 1s) running from the top-left towards the bottom-right. The text "Input Kludge" is positioned at the top center in a large, bold, black sans-serif font.

# Input Kludge

- ◆ **Исток**
  - Программное обеспечение не выдерживает входные тесты на ввод данных
- ◆ **Причина**
  - Алгоритмы не выявляют недопустимые комбинации символов, либо не обращают на них внимание вообще

# Input Kludge

## ◆ Как избежать

- Использовать алгоритмы, выполняющие лексический анализ
- Использовать регулярные выражения
- Использовать контекстно-свободные грамматики

# Walking through a Minefield



# Walking through a Minefield

## ◆ Исток

- Программное обеспечение содержит скрытые, трудновыявляемые дефекты
- Дефекты могут быть взаимосвязаны

# Walking through a Minefield

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Walking through a Minefield

## ◆ Как избежать

- Применение технологии разработки через тестирование (подробно см DEV-009)
- Постоянное выполнение различного рода тестирования
  - ◆ **Модульное**
  - ◆ **Интеграционное**
  - ◆ **Функциональное**
  - ◆ ...

# Copy-And-Paste Programming



# Copy-And-Paste Programming

- ◆ **Исток**
  - Копирование исходного текста понимается как повторное использование программного кода

# Copy-And-Paste Programming

## ◆ Симптомы и последствия

- Ошибка повторяется во многих местах программного кода
- Увеличение размера кода без общей производительности
- Код не может быть повторно использован
- Каждое исправление ошибки в коде уникально и нет общего метода по устранению этой ошибки

# Copy-And-Paste Programming

## ◆ Причины

- Нежелание тратить усилия на создание повторно используемого кода
- Организация не поддерживает повторное использование кода, а сосредоточена лишь на сроках проектов
- Организация считает, что код должен идеально подходить для новой задачи, прежде чем быть повторно используемым
- Повторно используемые компоненты были кем-то созданы, но не задокументированы
- Люди не знакомы с новой технологией и/или инструментом

# Copy-And-Paste Programming

## ◆ Исключение

- Когда требуется получить рабочий код быстро, как только возможно
- Цена заплаченная за такой код может оказаться высокой!

# Copy-And-Paste Programming

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?



# Copy-And-Paste Programming

- ◆ **Решение**

- Выполнение рефакторинга (выделение базового класса, выделение метода)

# Copy-And-Paste Programming

- ◆ Связанные антипаттерны
  - Spaghetti Code

# Error hiding/Exception handling



# Error hiding/Exception handling

- ◆ **Исток**

- Скрытие ошибок посредством применения нулевых обработчиков событий

```
try
{
}
catch(...)
{
}
```

# Error hiding/Exception handling

## ◆ Причины и последствия

- Повсеместное непродуманное использование обработчиков исключительных ситуаций маскирующих ошибку
- Функциональность системы падает молча, не оставляя понять шансов, что происходит

# Error hiding/Exception handling

- ◆ Сталкивались ли вы с данным антипаттерном ?
- ◆ Как вы решили эту проблему?

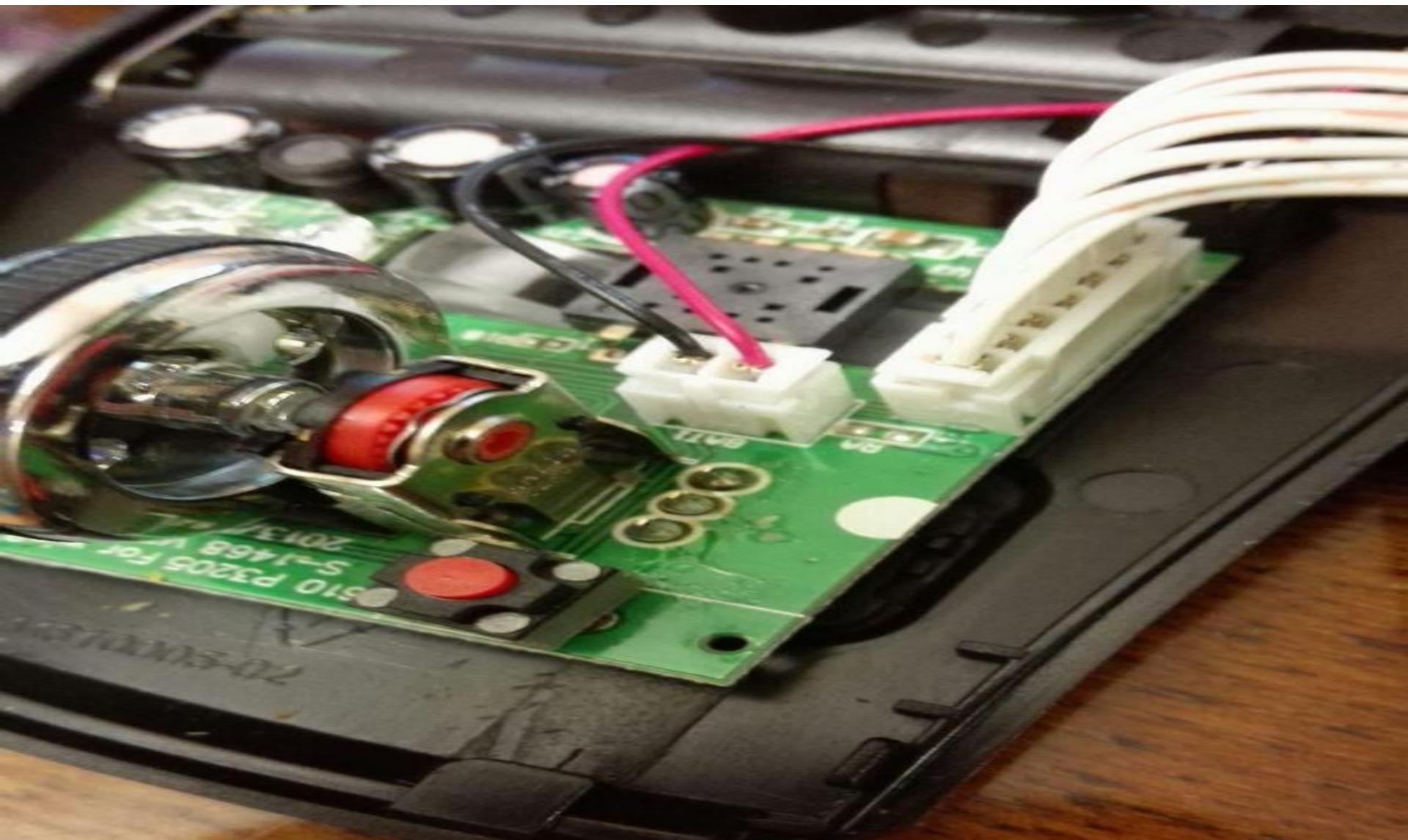


# Error hiding/Exception handling

## ◆ Решение

- Создать собственную иерархию исключений, классифицирующую возможные ошибки с точки зрения предметной области
- Выбрасывать исключение точно описывающее ошибку
- Обработчики исключений должны следовать от частного к общему
- Применять обработчик по умолчанию только в стартовом коде, для обработки прочих ошибок с целью «не пугать» пользователя «страшными сообщениями» программы

# Волшебная кнопка



# Волшебная кнопка

- ◆ **Симптомы**

- Смешение бизнес-логики со слоем презентации

- ◆ **Причина**

- Лень
  - Непродуманная архитектура
  - Быстро сделанное приложение «на коленке»

# Волшебная кнопка

- ◆ Сталкивались ли вы с анти-паттерном «Волшебная кнопка»?
- ◆ Как вы решили эту проблему?



# Демонстрационный пример

- ◆ Волшебство кнопки...



# Волшебная кнопка

## ◆ Решение

- Выполнить рефакторинг приложения, отделив бизнес-логику от слоя презентации
- Убрать дублирование кода

# Антипаттерны в управлении разработкой ПО

Модуль 4



**LXFT**  
**LISTED**  
**NYSE**

# Антипаттерны в управлении разработкой ПО

- ◆ **Дым и зеркала**
- ◆ **Раздувание программного обеспечения**
- ◆ **Функции для галочки**
- ◆ **Паралич от анализа**
- ◆ **View graph engineering**
- ◆ **Death By Planning**
- ◆ **Corncobs**
- ◆ **Fire Drill**



# Дым и зеркала



МЕНЮ

Gigantos  
TM

# Дым и зеркала

## ◆ СИМПТОМЫ

- Руководящая команда, которая ищет новые возможности для бизнеса поощряет ошибочное восприятие, что демонстрационная система показывает все ее возможности и качество конечного продукта, и даёт обещания, которые не соответствуют их технологиям



# Дым и зеркала

## ◆ Следствие

- Разработчики вынуждены обеспечивать обещанный функционал
- Пользователь не получает конечного продукта с заявленными функциями, в обещанные сроки и по обещанной цене

# Дым и зеркала

- ◆ Сталкивались ли вы с анти-паттерном «Дым и зеркала»?
- ◆ Как вы решили эту проблему?



# Дым и зеркала

- ◆ Решение
  - Правила
    - ◆ **готовая система стоит в три раза больше, чем экспериментальный образец**
    - ◆ **разработка любого ПО займет в два раза больше времени и будет стоить в два раза больше, чем ожидалось**
  - Экспертиза
    - ◆ **Квалифицированная оценка опытного специалиста**
  - Юридически
    - ◆ **В договор/соглашение добавить пункт об отсутствии юридически значимой гарантии, что продукт будет демонстрировать полезный функционал**

# Раздувание программного обеспечения



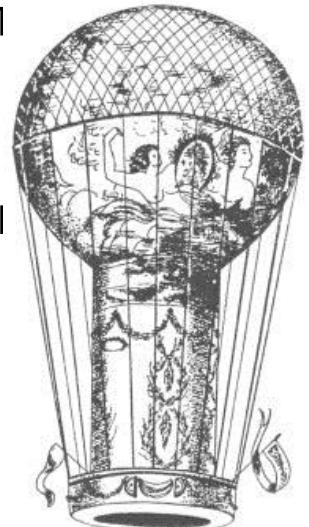
# Раздувание программного обеспечения

## ◆ СИМПТОМЫ

- Разрешение последующим версиям программного обеспечения требовать все больше и больше ресурсов

## ◆ Причины

- Добавление в программное обеспечение все новых и новых функций, не являющихся необходимыми для работ
- Использование технологий, значительно потребляющих ресурсы, для задач, где эти ресурсы не требуются



# Раздувание программного обеспечения

## ◆ Следствие

- Неоправданный рост требований к аппаратному обеспечению
- Экономические издержки
- Снижение качественных показателей, например быстродействие

# Раздувание программного обеспечения

## ◆ Решение

- Больше уделять внимание оптимизации программного обеспечения
- Избавление от неиспользуемых или редко используемых частей системы (лодочный якорь)
- Рефакторинг

# Функции для галочки



# ФУНКЦИИ ДЛЯ ГАЛОЧКИ

## ◆ СИМПТОМЫ

- Наспех сделанные функции, потому что они указаны в рекламном проспекте программного обеспечения

## ◆ Причины

- Добавление новых функций и возможностей, в том числе, не являющихся абсолютно необходимыми для работы



# Функции для галочки

- ◆ Сталкивались ли вы с анти-паттерном «Функции для галочки»?
- ◆ Как вы решили эту проблему?



# Analysis Paralysis



# Analysis Paralysis

## ◆ Исток

- Объектно-ориентированный анализ фокусируется на декомпозиции проблем с целью снизить их сложность
- Очень часто разработчики уходят с абстрактного уровня на уровень реализации, что делает анализ предметной области невозможным
- Эксперты предметной области перестают понимать предложенную им модель для экспертизы

# Analysis Paralysis

## ◆ Симптомы и последствия

- Из-за кадровых изменений проект постоянно перезапускается или постоянно меняется направление проекта
- Вопросы реализации постоянно возникают на этапе анализа
- Стоимость анализа превышает ожидания без предсказуемого конечного результата
- Имплементация, например использование GoF паттернов обсуждается на этапе объектно-ориентированного анализа

# Analysis Paralysis

- ◆ Сталкивались ли вы с анти-паттерном «Analysis Paralysis»?
- ◆ Как вы решили эту проблему?

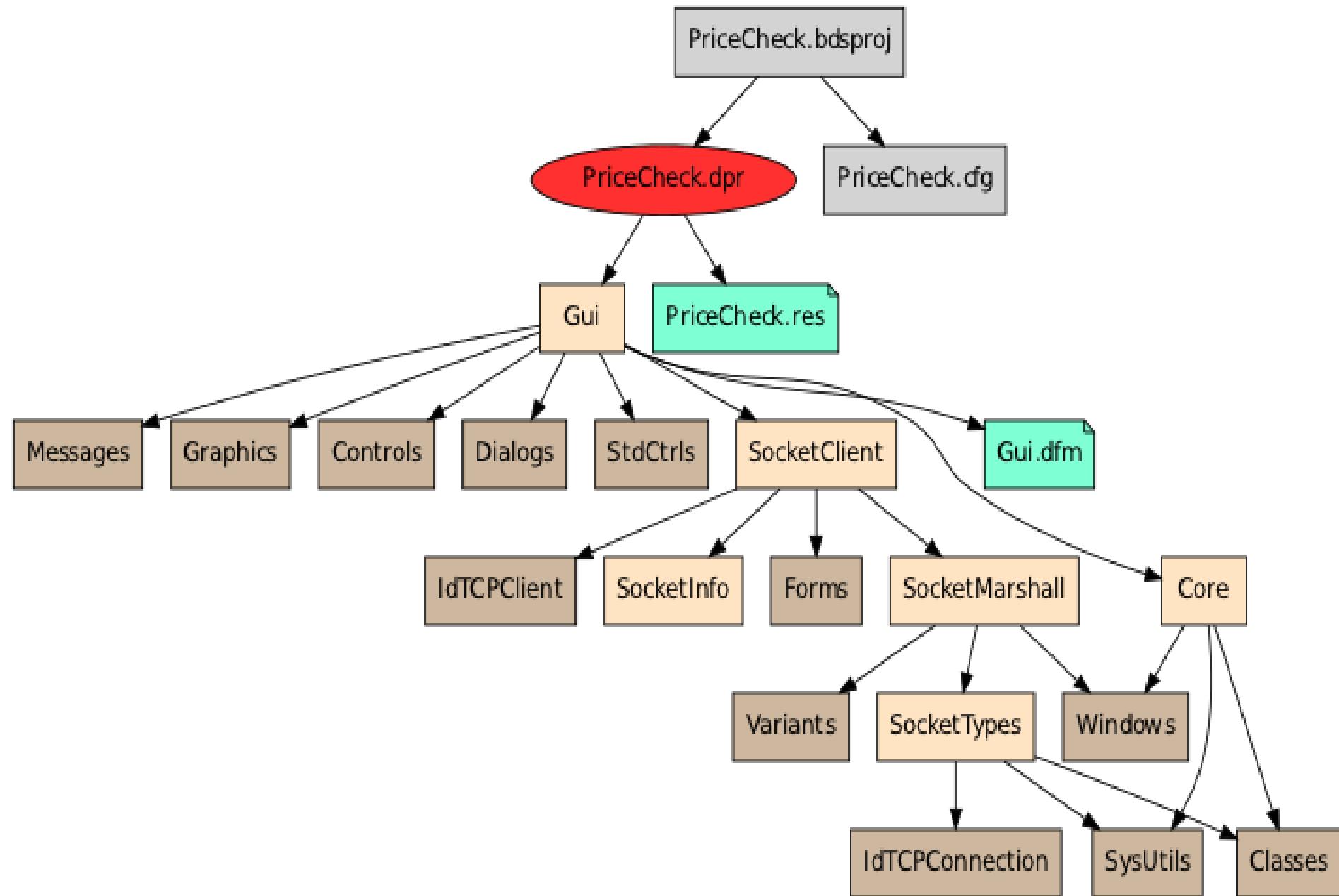


# Analysis Paralysis

## ◆ Решение

- Первоначальный объектно-ориентированный анализ включает в себя обзор системы высокого уровня
- Инкрементальная разработка, когда уровень абстракции понижается постепенно на каждом этапе

# ViewGraph Engineering



# ViewGraph Engineering

## ◆ Исток

- Многие разработчики уделяют слишком много внимания документации в противовес разработке программного обеспечения
- Многие модели, построенные для документации не отражают реальной картины и являются псевдо-схемами

# ViewGraph Engineering

- ◆ Сталкивались ли вы с анти-паттерном «ViewGraph Engineering»?
- ◆ Как вы решили эту проблему?



# ViewGraph Engineering

## ◆ Как избежать

- Направить разработчиков на построение протипов вместо построения абстрактных моделей
- Прототипы могут ответить на технические вопросы, которые не могут быть получены из анализа
- Прототип помогает сократить обучение новым технологиям

# ViewGraph Engineering

- ◆ **Как избежать**
  - Прототипы бывают двух видов:
    - ◆ **Макеты**
    - ◆ **Инженерные**

# Death By Planning



# Death By Planning

## ◆ Исток

- Слишком детальное планирование, содержит множество хаотичных действий, которые не обязательно для программных проектов
- Смерть от планирования происходит, когда большую часть проекта занимает планирование, что в конечном итоге приводит к прекращению проекта из-за больших затрат

# Типы сверхпланирования

- ◆ **Glass Case Plan**
  - Планирование прекращается как только проект начинается
- ◆ **Detailitis Plan**
  - Планирование продолжается до тех пор, пока проект не перестанет существовать по какой-либо причине

# Death By Planning

## ◆ Симптомы и последствия (Glass Case Plan)

- Неумение планировать на прагматическом уровне
- Фокус на затраты, а не на разработку
- Кризис управления проектом
- Отмена проекта
- Потеря ключевых сотрудников

# Death By Planning

## ◆ Симптомы и последствия (Detailitis Plan)

- Неумение планировать на прагматическом уровне
- Фокус на затраты, а не на разработку
- Тратится больше времени на планирование, описание прогресса, чем на разработку программного обеспечения
- Лидеры команд планируют активности команды и активности ее членов
- Разбивание всех активностей разработчиков на задачи

# Death By Planning

- ◆ Сталкивались ли вы с анти-паттерном «Death By Planning»?
- ◆ Как вы решили эту проблему?



# Death By Planning

## ◆ Решение

- Планы должны показать конечные результаты
- Показатели должны быть определены на двух уровнях:
  - ◆ **Продукты**
  - ◆ **Компоненты (в продуктах)**

# Corncobs



# Corncobs

## ◆ Исток

- «Трудные люди», которые усугубляют проблемы проекта из-за индивидуальных аспектов личности, но зачастую трудности возникают из-за личных мотивов для признания или денежного стимулирования
- При общении с такими людьми важно помнить, что эти люди больше внимания уделяют политике, чем технологии. Они обычно являются экспертами манипулятивной политике и технически-ориентированные люди сами того не желая могут стать их жертвой

# Corncobs

## ◆ Причины и последствия

- Команда разработки или проекта не может достичь прогресса, потому что кто-то не согласен с их ключевой целью или основными процессами и постоянно пытается их изменить
- Один человек постоянно высказывает возражения под видом беспокойства о производительности, о надежности и т.д., которые являются неразрешимыми
- Часто «кукурузник» это менеджер, который не находится под непосредственным руководством старшего программного менеджера или менеджера проекта
- Компания не имеет четкого процесса принятия решений, что позволяет руководителю к некстати вмешиваться, выходя за рамки своих полномочий

# Corncobs

- ◆ Сталкивались ли вы с анти-паттерном «Corncobs»?
- ◆ Как вы решили эту проблему?



# Corncobs

- ◆ **Тактические решения**

- Передайте ответственность
- Изолировать проблему
- Задать вопрос для определения личной позиции «кукурузника»

# Corncobs

- ◆ **Оперативные решения**

- Проведите корректирующее интервью
- Дружелюбный перевод сотрудника. С помощью Head Hunter подобрать человеку более подходящую для него должность в организации

# Corncobs

- ◆ **Стратегические решения**

- Сделать одну группу из всех «кукурзников», чтобы они сами боролись друг с другом за влияние
- Сделать «пустой отдел» - сотрудник является единственным сотрудником своего же отдела
- Ликвидация силой – когда нет другого выхода

# Corncobs

## ◆ Известные исключения

- В проектах, где задействовано несколько организаций полезно иметь «кукурузника», чья роль заключается в защите существующей архитектуре от изменений, если есть какие-либо противоречия или несколько способов этого изменения
- Когда организация мирится по каким-либо причинам с действиями «кукурузника»

# Fire Drill



# Fire Drill

## ◆ Исток

- Проект инициируется, но разработка задерживается на несколько месяцев из-за технических и политических вопросах, которые решаются на уровне руководства
- Менеджмент предотвращает развитие проекта разработчиками ,просьбами подождать или давая противоречивые направления
- Через несколько месяцев руководство требует быстрого прогресса проекта, что ведет, в конечном итоге, к его отмене

# Fire Drill

- ◆ Сталкивались ли вы с анти-паттерном «Fire Drill»?
- ◆ Как вы решили эту проблему?



# Fire Drill

## ◆ Решение

- Управление проектом отвечает за разработку продукта независимо от нерешенных управлеченческих проблем
- Использовать разработку управляемой архитектурой

# **Подведение итогов**

# Вопросы?

## Внутренние ресурсы

Расписание,  
курсы,  
тренеры

[IntHR](#)  
[Lux Town](#)

Условия обучения,  
логистика,  
контакты

[Lux Town](#)

## Внешний ресурс



### Информация об учебном центре

[www.luxoft-training.ru/about](http://www.luxoft-training.ru/about)

### Расписание

[www.luxoft-training.ru/timetable](http://www.luxoft-training.ru/timetable)

### Каталог курсов

[www.luxoft-training.ru/training/catalog\\_directions](http://www.luxoft-training.ru/training/catalog_directions)

### Контакты

[www.luxoft-training.ru/contacts](http://www.luxoft-training.ru/contacts)



[www.facebook.com/TrainingCenterLuxoft](http://www.facebook.com/TrainingCenterLuxoft)