

# Переписать нельзя рефакторить!

Пыхтин Павел  
Singularis Lab, Ltd.



# LEGACY CODE

Welcome to the project!  
Here's the codebase.

[seanthelexguy.com/blog](http://seanthelexguy.com/blog)

# Legacy

- Уже работает в Production
- Большое количество заплат
- Устаревшие технологии
- Неактуальная документация
- Отсутствие единого стиля
- ...

Технический  
долг

# Выплачиваем долги

- Переписать с нуля
  - Избавляемся от всех ошибок
  - Получаем свободу в выборе архитектуры и технологий
  - Долго
  - Несет заметные убытки для бизнеса
  - «В этот раз уж точно получится!»



# Выплачиваем долги

- Рефакторить модуль за модулем
  - Сложно
  - Приходится считаться с унаследованной архитектурой (даже если ее нет)
  - Возможно окажется еще дольше
  - Приложение продолжает развиваться
  - Стоимость распределена по времени
  - Есть возможность учесть новые требования бизнеса



# Рефакторинг глазами заказчика

- «Мы сделали плохо, заплатите нам, чтобы мы все исправили»
- «Никаких изменений вы не заметите, но внутри будет намного лучше»
- «Я вчера прочитал, что наш фреймворк уже два месяца как устарел. Надо все переписать»



# Как убедить заказчика?

- У инженера и руководителя разные ценности
- Ваши действия:
  - Посмотрите на проблему с точки зрения бизнеса
  - Оцените текущие затраты на поддержку кода
  - Постарайтесь оценить эффект от рефакторинга
  - Составьте план работ с указанием требуемого времени и ожидаемых результатов
  - Объясните причины, по которым возникли проблемы требующие рефакторинга

Вид работ	Объем работ	Ожидаемый эффект	Приоритет
Рефакторинг модуля импорта	80	Повышение надежности, уменьшение числа возникающих ошибок и, как следствие, повышение лояльности пользователей	Низкий
Замена фронтенд фреймворка	160	Уменьшение времени загрузки страницы на 20%, упрощение поддержки и доработки	Высокий
...	...	...	



# С чего начать?

- Мониторинг, логирование
- Автоматизация процессов
- Документация

# Мониторинг и логирование

- Упрощает решение имеющихся и новых проблем
- Дает представление о текущем состоянии проекта
- Обеспечивает быстрое оповещение о возникающих проблемах
- Можно использовать в качестве формального показателя при оценке выгоды от рефакторинга

# Автоматизация процессов

- Сокращает издержки
- Позволяет формализовать процессы
- Позволяет выявить проблемные места в системе
- Что автоматизировать:
  - Сборку
  - Развертывание
  - Тестирование
  - ...
  - Все, что занимает время, но не требует вашего полного внимания

# Документация

- Сбор и актуализация требований
- Удаление избыточных требований
- Написание сценариев тестирования

# Приступаем

- Всегда оставляйте после себя код чище, чем он был
- Вся команда должна быть вовлечена в процесс
- Используйте инструменты

# Статический анализ

- Позволяет отслеживать динамику состояния кода
- Позволяет выявлять проблемные места и цели для последующего рефакторинга



# Тесты

- Несколько тестов лучше, чем ничего
- Тесты сами по себе являются документацией
- Тесты помогут лучше спроектировать архитектуру

# Практики

- Extract Method
- Inline Method
- Inline Temp
- Replace Temp with Query
- Introduce Explaining Variable
- Split Temporary Variable
- Remove Assignments to Parameters
- Replace Method with Method Object
- Substitute Algorithm
- Move Method
- Move Field
- Extract Class
- Inline Class
- Hide Delegate
- Remove Middle Man
- Introduce Foreign Method
- Introduce Local Extension
- Self Encapsulate Field
- Replace Data Value with Object
- Change Value to Reference
- Change Reference to Value
- Replace Array with Object
- Duplicate Observed Data
- Change Unidirectional Association to Bidirectional
- Change Bidirectional Association to Unidirectional
- Replace Magic Number with Symbolic Constant
- Encapsulate Field
- Encapsulate Collection
- Replace Record with Data Class
- Replace Type Code with Class
- Replace Type Code with Subclasses
- Replace Type Code with State/Strategy
- Replace Subclass with Fields
- Decompose Conditional
- Consolidate Conditional Expression
- Consolidate Duplicate Conditional Fragments
- Remove Control Flag
- Replace Nested Conditional with Guard Clauses
- Replace Conditional with Polymorphism
- Introduce Null Object
- Introduce Assertion
- Rename Method
- Add Parameter
- Remove Parameter
- Separate Query from Modifier
- Parameterize Method
- Replace Parameter with Explicit Methods
- Preserve Whole Object
- Replace Parameter with Method
- Introduce Parameter Object
- Remove Setting Method
- Hide Method
- Replace Constructor with Factory Method
- Encapsulate Downcast
- Replace Error Code with Exception
- Replace Exception with Test
- Pull Up Field
- Pull Up Method
- Pull Up Constructor Body
- Push Down Method
- Push Down Field
- Extract Subclass
- Extract Superclass
- Extract Interface
- Collapse Hierarchy
- Form Template Method
- Replace Inheritance with Delegation
- Replace Delegation with Inheritance
- Tease Apart Inheritance
- Convert Procedural Design to Objects
- Separate Domain from Presentation
- Extract Hierarchy

# Практики

- Выделение интерфейсов
- Паттерн «Душитель»
- Агрегация предыдущей реализации

# Выделение интерфейсов

- Цель:
  - Уменьшить связность классов
- Методика:
  - Группировка переменных и методов в соответствии со сценариями использования
  - Инкапсуляция в отдельный класс
  - Перемещение ответственности за создание класса в контейнер инъекции зависимостей

# Агрегация предыдущей реализации

- Цель:
  - Обеспечить возможность постепенного рефакторинга имеющейся реализации
- Методика:
  - Выделить интерфейс существующей реализации
  - Создать новую реализацию с тем же интерфейсом, агрегирующую существующую в качестве скрытого поля
  - Пробросить вызовы в старую реализацию
  - Новые функции реализуются в новом сервисе
  - Старые функции рефакторятся и переносятся в новую реализацию по мере возможности

# Паттерн «Душитель»

- Цель
  - Обеспечить возможность постепенного рефакторинга имеющейся реализации
- Методика:
  - Создается новое приложение/сервис, которое впоследствии заменит существующее приложение
  - Новые функции реализуются в новом приложении
  - Старые функции переносятся в новое приложение по мере возможности





# To be continued...

# Наши инструменты

- Team City
- Sonar Qube
- BackstopJS
- Eslint
- Test link

# Байки

- Рефакторинг кода после 2-х лет разработки фрилансерами
  - Занял примерно полтора года
  - Средний размер класса сократился с 1000 строк до 200
  - Исправлено большое количество трудно воспроизводимых ошибок
  - Функциональность системы была расширена вдвое

# Байки

- Переход с ADO.net на Nhibernate
  - Занял около полугода
  - Решены проблемы кэширования
  - Время старта приложения уменьшилось с 20 секунд до 7 секунд

# Байки

- Переход с jQuery + DevExpress на AngularJS
  - Около 15 000 строк JS кода
  - Выполнен на 50%
  - Трудозатраты на реализацию новой функциональности сократились вдвое

# Памятка рефакторинга

- Определить масштаб проблем
- Составить план рефакторинга (сроки, трудозатраты, ожидаемый результат)
- Настроить инструменты (логи, тесты, статический анализ...)
- Донести до команды ценность рефакторинга и качественного кода
- ...
- Profit!



**I HAVE A DREAM**



**THAT ONE DAY ALL LEGACY CODE  
WILL BE GONE**

[memegenerator.net](http://memegenerator.net)

# Спасибо за внимание!

Павел Пыхтин

- [pykhtin.pavel@singularis-lab.com](mailto:pykhtin.pavel@singularis-lab.com)



<https://www.singularis-lab.com/>



<https://www.linkedin.com/company/singularis-lab-llc>



<http://habrahabr.ru/company/singularis>



[http://vk.com/singularis\\_lab](http://vk.com/singularis_lab)

# Ссылки

- <https://www.sitepoint.com/technical-debt-rescuing-legacy-code-through-refactoring/>
- <http://blog.byndyu.ru/2014/01/blog-post-8.html>

# Технический долг

- Метафора, описывающая проблемы архитектуры или реализации.
- Не обязательно свидетельствует о низкой квалификации разработчика
- Стоит ли осознанно влезать в долги:
  - Это жизненно важно для проекта (прототип для демонстрации, проверка гипотезы, hotfix...)
  - Поддержка текущей реализации не планируется (никем и никогда)