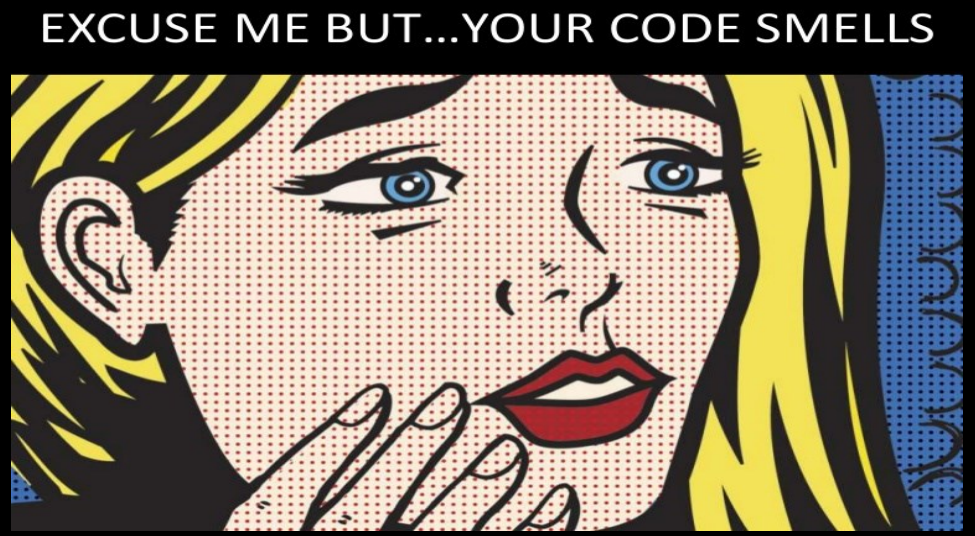
Лош програмен код

Или кога е наложителна преработка на кода



Учителски екип

Обучение за ИТ кариера

https://it-kariera.mon.bg/e-learning/

Съдържание

- Видове лош програмен код:
  - Разводняване (bloaters)
  - Неясноти (obfuscators)
  - Злоупотреби на ООП (OO abusers)
  - Блокажи за промени (change preventers)
  - Излишества (dispensables)
  - Сглобки (couplers)

Лош програмен код (Code Smells)

- Лош програмен код == определени структури в кода, които подсказват, че е наложителна преработка
- Видове лош програмен код:
  - Разводняване (bloaters)
  - Неясноти (obfuscators)
  - Злоупотреби на ООП (OO abusers)
  - Блокажи за промени (change preventers)
  - Излишества (dispensables)
  - Сглобки (couplers)

https://sourcemaking.com/refactoring/smells

Лош код: Разводнявания (Bloaters)

- Дълги методи
  - По-добре е методите да са по-къси (по-лесно именуване, по-разбираеми са, по-малко повторения на код)
- Големи класове
  - Твърде много променливи в екземплярите
  - Нарушават принципа „Една-едничка цел" (Single Responsibility)
- Мания за прости данни (за прекаленото им използване)
  - Прекаляване с прости типове, вместо добра абстракция
  - Може да бъдат обособени в свой клас с вградено валидиране

Лош код: Разводнявания (2)

- Дълъг списък с параметри (`in` / `out` / `ref`)
  - Може да подсказва процедурно вместо ОО програмиране
  - Може би методът върши твърде много неща
- Групички от данни
  - Набор от данни, винаги използвани заедно без да са групирани
  - Например полетата на кредитна карта в `Order` класа
- Избухване в комбинации
  - Напр. `ListCars()`, `ListByRegion()`, `ListByManufacturer()`, `ListByManufacturerAndRegion()` и т.н.
  - Решение може да е шаблона Interpreter (LINQ)

Лош код: Разводнявания (3)

- Чудати решения
  - Странни решения на често срещани проблеми
  - Непоследователност
  - Решение: Заменете алгоритъма или използвайте Adapter
- Клас, който нищо не прави
  - Решение: Слейте го с друг клас или го премахнете
- Задължителен начален / завършващ програмен код
  - Изисква винаги няколко реда код, преди да се ползва
  - Решение: параметър-обект, factory метод, `IDisposable`

Лош код: Неясноти (Obfuscators)

- Обхват
  - Целта на кода е неясна и се нуждае от коментар (лошо)
  - Програмният код е твърде дълъг, за да е обозрим (лошо)
  - Решение: частичен клас, нов клас, подреждане на кода
- Коментари
  - Трябва да обясняват **ЗАЩО**, а не **КАКВО** или **КАК**
  - Добрите коментари: дават повече информация, препратки към ресурси, обясняват алгоритъм, причини или контекст
  - Препратка: Смехотворни коментари

Лош код: Неясноти (2)

- Къси / неподходящи имена
  - Трябва да са подходящи, говорвящи и еднотипно използвани
- Вертикално отделяне
  - Трябва да декларирате променливите точно преди първата им употреба, за да се избегне скролирането
  - Или използвайте малки функции
- Непоследователност
  - Следвайте POLA (Принципа на най-малкото чудене)
  - Непоследователността е объркваща и разсейваща
- Неясни намерения
  - Кодът трябва да е толкова обяснителен, колкото е възможно

Лош код: Злоупотреби на ООП (OO Abusers)

- Използване на switch израз с обекти
  - Може да бъде заменено с полиморфизъм
- Временни полета
  - Когато се предават данни между методи
- Клас, зависещ от подклас
  - Класовете не могат да бъдат разделени (взаимозависими са)
  - Може да наруши принципа за заместване на Лисков
- Неподходящо статично поле
  - Силно сдвояване между static и викащия го
  - Което е static, не може да се подмени или използва за друго

Лош код: Блокажи за промени (Change Preventers)

- Многократна промяна
  - Клас, който често се променя по различни начини / причини
  - Нарушава SRP принципа (single responsibility principle)
  - Решение: да се отдели класа
- Принудителни промени
  - Една промяна налага промени в много класове
    - Трудно е да бъдат намерени, лесно е да се пропусне някой
  - Решение: местене на методи и полета, подреждане на кода

Лош код: Блокажи за промени (2)

- Сложност на условията
  - Увеличава общата сложност (Cyclomatic complexity, броят на различните пътища, по които може да бъде изпълнен кода)
  - Симптоми: много влагания (arrow code) и бъгливи `if`-ове
  - Решение: отделяне на метод, шаблон „Strategy“, „State“ или „Decorator“
- Зле написани тестове
  - Лошо написаните тестове може да възпрепятстват промените
  - Много силни зависимости

Лош код: Излишества (Dispensables)

- Мързеливи класове
  - Класове, които не правят достатъчно, за да оправдаят своето съществуване трябва да бъдат премахнати
  - Всеки клас иска време и усилие да бъда разбран и поддържан
- Класове с данни
  - Класове само с полета и свойства
  - Липсващо валидиране? Програмният код е в други класове?
  - Решение: да се премести свързаната с данните логика в класа

Лош код: Излишества (2)

- Повторения на код
  - Нарушава DRY принципа
  - Резултат е от copy-pasted код
  - Решение: отделяне на метод, клас, pull-up метод, шаблон Template Method
- Ненужен код (такъв, който никога не се ползва)
  - Обикновено се открива от инструментите за статистически анализ
- Спекулативни обобщения
  - „Някой ден може да ни потрябва …"
  - YAGNI принципа

Лош код: Сглобки (Couplers)

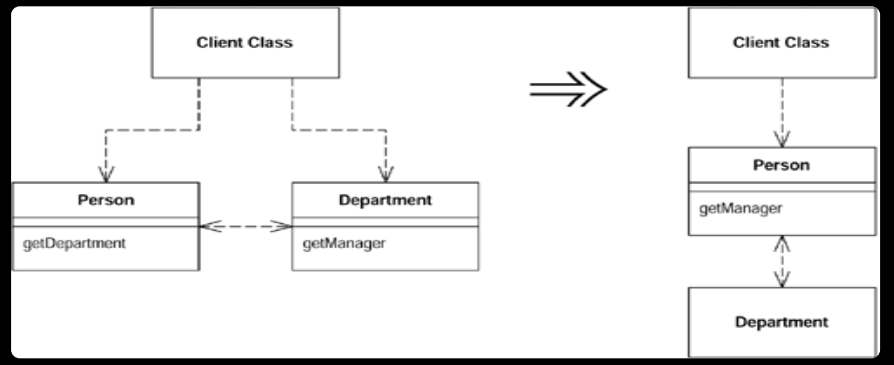
- Завист за чужди екстри (Feature envy)
  - Метод, който изглежда по-заинтересуван от друг клас, различен от този, в който е метода
  - Дръжте заедно нещата, които взаимно се променят
- Неуместно интимничене (Inappropriate intimacy)
  - Класове, които знаят твърде много един за друг
  - Проблеми: при наследяване, двупосочна връзка
  - Решение: местете на методи / полета, извличане на клас, промяна на връзката в еднопосочна, делегиране вместо наследяване

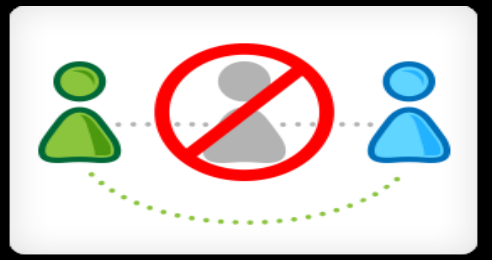
Лош код: Сглобки (2)

- Законът на Деметра (Law of Demeter, LoD)
  - Даден обект трябва да очаква възможно най-малко определена структура на свойства или каквото и да е
  - Лош пример: `customer.Wallet.RemoveMoney()`
- Индиректно разкриване (Indecent exposure)
  - Някои класове или членове са public без да е нужно
  - Нарушава капсулирането
  - Може да доведе до неуместно интимничене

Лош код: Сглобки (3)

- Верижни съобщения
  - `Something.Another.SomeOther.Other.YetAnother`
  - Тясна зависимост между клиент и структурата за достъп
- Посредник
  - Делегиране, отишло твърде далеч
  - Понякога може да го премахнем или вградим
- Скитащи данни
  - Да се предават данни само защото са нужни на някой друг
  - Решение: Премахваме посредническите данни, извличаме клас





Лош код: Сглобки (4)

- Изкуствени зависимост (Artificial coupling)
  - Неща, които не зависят едно от друго, няма нужда изкуствено да се обвързват в зависимост
- Скрита временна зависимост (Hidden temporal coupling)
  - Не трябва да предполагаме реда на операциите
  - Например класът pizza не трябва да знае стъпките за правене на пица -> Template Method шаблон
- Скрити зависимости (Hidden dependencies)
  - Класовете трябва да обявяват своите зависимости още в конструктора си
  - **new** е връзката / принципа на Dependency Inversion

Обобщение

- Видове лош програмен код:
  - Разводняване (bloaters)
  - Неясноти (obfuscators)
  - Злоупотреби на ООП (OO abusers)
  - Блокажи за промени (change preventers)
  - Излишества (dispensables)
  - Сглобки (couplers)

Лош програмен код

Въпроси?

case

Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист"

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под свободен лиценз **CC-BY-NC-SA**