Рефакториране

Означава да променим дизайна, структурата на един правилно работещ код, без да променяме неговото поведение. Но понякога се налага да променим и самата функционалност (поведението) на кода - например както BG coder е работил само със стрингове, след това кода е бил рефакториран така че да работи и с файлове

When to refactor:

* To make adding a new function easier – за това е хубаво класовете да depend-ват на външни зависимости, а не на конкретни имплементации
* As part of the process of fixing bugs
* When reviewing someone else’s code
* When doing test-driven development

Когато пишем unit тестове, правилното им изпълнение гарантира, че не сме променили поведението на кода. Ако няма unit тестове, трябва да напишем такива, преди да започнем да рефакторираме.

Code smells == certain structures in the code that suggest the possibility of refactoring:

1. Код, който е много дебел, но върши малко работа

* Дълги методи – за да се реши този проблем, конкретният дълъг метод се нацепва на по- малки методи
* Дълги класове – когато има прекалено много методи или instance променливи в кода
* Когато има употреба на прекалено много примитивни типове данни, т.е. ако един метод приема много параметри (напр. име, парола, username и т.н.). Вместо това може всички тези данни да ги групираме в един клас.

1. Когато самият код е толкова заплетен, че нямаме идея какво става

* Коментарите трябва да отговарят на въпроса **защо** дадено парче код се използва, а не какво и как го прави
* Лошо именуване на идентификаторите

1. Когато е вкарано прекалено много обектно-ориентирано програмиране (напр. когато даден клас има само едно поле с данни – най-вероятно няма смисъл)

* Ако даден клас се използва само и единствено от друг клас, то първия клас е добре да го направим вътрешен за втория клас
* Не трябва да пишем дадено парче код само защото „един ден може да ни потрябва“
* Класовете не бива да expose- ват навън неща, които не се използват публично. Т.е. трябва максимално да намаляваме scope- а на мембърите на класа

Лоши практики: bloaters (когато излишна част от функционалността на даден клас е разкрита за външно използване), object-oriented abusers (прекомерно използване на ООП, особено при прости решения на дадени проблеми), long parameter lists, combinatorial explosion (e.g. ListCars(), ListByRegion(), ListByManufacturer(), etc.), oddball solution (когато знаем определен, работещ начин за решаване на даден проблем, но ние се опитваме да измислими нов, защото предишния не е „идеален“), regions, inappropriate static fields

Много пъти това което рефакторираме го решаваме чрез инстинкт, т.е. някои практики са правилни за следване в едни случаи, а в други не

Понякога може да използваме Lambda изрази вместо да извеждаме даден код в отделен метод

Вместо да правим един интерфейс с 5 метода, а ние искаме да използваме само 3 от тези 5 метода, е добре да направим 2 интерфейса, единия с 3 метода и другия да го наследява и да има още 2 метода

Refactoring: the typical process

1. Save the code you start with (backup)
2. Make unit tests to assure the code’s behavior is correct after the refactoring
3. Do refactoring one at a time
4. Run the tests and they should pass
5. Check-in (т.е. да push- нем готовия код)

**Общо взето когато кода не следва практиките за качествен код, то той трябва да бъде рефакториран**