Refactoring

Mikael Svahnberg*

2023-02-01

Contents

1 What is Refactoring **1**

2 Refactoring Checklist **1**

3 Clean Code **2**

4 Dirty Code – Technical Debt **2**

5 Avoiding Technical Debt **3**

6 Code Smells **3**

7 When to Refactor **4**

8 Refactoring Techniques **4**

1 What is Refactoring

Systematic process to *improve* the code without changing its functionality.

- Improve readability (e.g. rename variables)
- Improve maintainability (e.g. restructure design)

2 Refactoring Checklist

☐ You leave the code cleaner than you found it.

☐ There is no new functionality added together with a refactoring

☐ All existing tests still pass after refactoring

*Mikael.Svahnberg@bth.se

3 Clean Code

☐ Obvious to other programmers

- Reasonable size of each component/class/method
- Good names for classes attributes, methods, variables

☐ Does not contain duplicated code

☐ Does not contain unnecessary classes

☐ Passes all tests

→ cheaper to maintain

4 Dirty Code – Technical Debt

(The term *Technical Debt* is coined by Ward Cunningham)

- Quick fixes that we promise to ourselves that we are going to fix later.
- Goldplating that has no use right now but which we still need to maintain.
- Business Pressure
  - Ship it! We'll fix the details later...
  - Botch it so it more or less works. Hide the parts that are "under construction"
- Lack of Understanding of Consequences of Technical Debt
  - Technical Debt cost in time and resources for *all* development.
- Lack of adherence to original design decisions
  - maybe forgotten?
  - architecture decay
  - brittle system
  - changes in one component/class affect other classes
  - → time to trace and understand effect of changes
- Lack of documentation
  - How can you adhere to orignal design decisions if they are not documented?
  - Training of new employees
- Lack of interaction between team members
  - Shared understanding of design decisions means stricter adherence and less brittle systems
- Lack of tests
  - (Automated) tests bring direct feedback on what works or not.

5 Avoiding Technical Debt

- Maintain Updated Design Documentation
- Apply Refactoring Regularly
- Compliance Monitoring (are all design documents up to date, does the code adhere to guidelines, etc.)
- Training, e.g. on code standards, general competencies.

6 Code Smells

- Originally by Martin Fowler.
- Can be summarised as "You know all the bad things your OO design teacher told you not to do? Well, don't!"

Examples

Bloaters Code that grows too long → low cohesion

- long methods
- long parameter lists
- large classes
- using primitives instead of small objects

Object Orientation Abusers incorrect use of object oriented programming principles

- Classes that do the same things but with different method names
- Classes that only partially implement the interface from a superclass
- Long and elaborate switch statements rather than relying on polymorphism

Change Preventers Responsibilities are scattered through code so you need to change several places at once

- When many methods need to be edited for a single change (e.g. adding a new product type)
- Parallel inheritance hierarchies: Adding a class in one hierarchy means you also need to add a class in another hierarchy

Dispensables Pointless code or text that does not contribute anything

- Too many comments
- Duplicate code
- Dead Code
- Classes that no longer do anything meaningful
- Adding classes or inheritance hierarchies for future needs

Couplers Things that tie classes too closely together

- Using data in other classes (more than your own data)
- Message chains `myFancyObject->getFrobnicator()->createFluxCapacitor()->initiate()`

7 When to Refactor

Rule of Three

1. First time, just get it done
2. Second time you do something similar, recognise that it is similar but do it anyway
3. Third time – refactor!

Adding a Feature

- Refactor while reading existing code – as a part of understanding ig

When fixing a bug

- Clean up the code while looking for the bug

Code Reviews

- Regular activity with the purpose of cleaning up the code

A Little Bit All The Time

- Make this part of your normal velocity
- Easier than trying to motivate code reviews to the powers that be

8 Refactoring Techniques

(A selection that has a design impact: There are many more techniques for how to write clearer code *within* methods)

- Break out code into new methods to simplify the code
- Move methods and attributes to the class that should be responsible for them
- Remove classes that do not have any responsibilities
- Hide delegates to avoid method chains. If you are just "object hopping" to reach the right object, then you know too much about the design.
- Use wrapper classes to add functionality to libraries.
- Use getters and setters to access data
- Keep code from different layers/components separate. Duplicate data that should pass between components.
- Introduce classes to maintain collections (xxxManager, xxxContainer, . . .)
- Use Design Patterns instead of if-then-else chains.
- Create methods for complex if-then-else statements: `if condition() then trueCondition() else falseCondition()`

- Always return a meaningful object (e.g. a Null Object)
- Rename classes/methods/parameters/attributes/variables to meaningful and readable names
- Separate queries from modifiers → avoid side effects in code
- Parameterise method (from `frobnicateA(), frobnicateB(), \dots` to `frobnicate(type)`
- Replace complex constructor with a factory method.
- Apply Design Patterns
- Apply Fundamental Object Oriented Design Principles