Programming Project:
Code Quality

Guillermo Román Díez

groman@fi.upm.es

E.T.S. Ingenieros en Informática
Universidad Politécnica de Madrid

2019-2020

Measuring code quality

Question

What is the *quality of a program*?

Measuring code quality

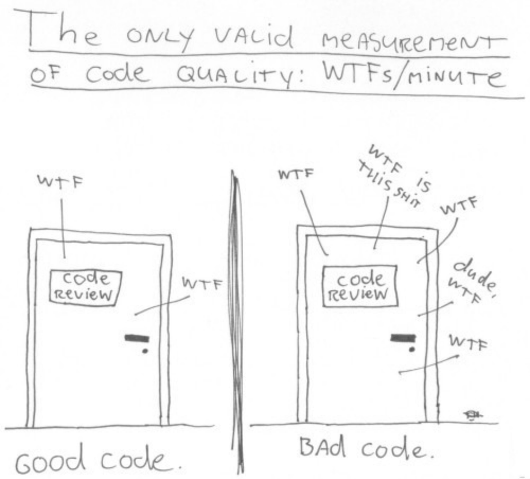
Question

What is the *quality of a program*?

Question

How can we measure the quality of a program?

Measuring code quality



(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

Taken from: `http://www.osnews.com/story/19266/WTFs_m`

High quality code

"We can say that the code is of high quality when productivity remains high in the presence of change in team and goals." [1]

[1] Ward Cunningham

Code quality

- After been a programmer for more than two or three years, it is very likely that you have been slowed down by messy code
- Any modification in messy code produces multiple errors in unexpected places
- After a while working on messy code, the productivity significantly decreases day by day
- The quality of the code is directly related to its **maintainability**

Clean code definitions

- Let us describe the main goal of a **good programmer**

[2] Martin Fowler
[3] Martin Golding

Clean code definitions

- Let us describe the main goal of a **good programmer**

  Any fool can write code that a computer can understand.
  Good programmers write code that humans can understand [2]

[2] Martin Fowler

[3] Martin Golding

Clean code definitions

- Let us describe the main goal of a **good programmer**

  Any fool can write code that a computer can understand.
  Good programmers write code that humans can understand [2]

- Let us continue with a **recommendation**:

[2] Martin Fowler

[3] Martin Golding

Clean code definitions

- Let us describe the main goal of a **good programmer**

  Any fool can write code that a computer can understand. Good programmers write code that humans can understand [2]

- Let us continue with a **recommendation**:

  Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live [3]

[2] Martin Fowler

[3] Martin Golding

Clean code definitions

- Let us describe the main goal of a **good programmer**

  Any fool can write code that a computer can understand. Good programmers write code that humans can understand [2]

- Let us continue with a **recommendation**:

  Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live [3]

- All programmer have their own idea of what is **clean code**
- Let us see some definitions written by deeply experienced programmers to figure out the most relevant ideas behind the notion of clean code

[2] Martin Fowler

[3] Martin Golding

Clean code definitions

Grady Booch [4]

Clean code is **simple** *and* **direct***. Clean code* **reads like well-written prose***. Clean code* **never obscures** *the designer's intent but rather is full of crisp abstractions and straightforward lines of control*

[4] One of the parents of the Unified Modeling Language and author of *Object Oriented Analysis and Design*

Clean code definitions

Dave Thomas [5]

Clean code **can be read, and enhanced** *by a developer other than its original author. It has* **unit and acceptance tests***. It has meaningful names. It provides* **one way** *rather than many ways for doing one thing. It has* **minimal dependencies***, which are explicitly defined, and provides a clear and* **minimal API***. Code should be literate since depending on the language, not all necessary information can be expressed clearly in code alone*

[5] Founder of Object Technology International and godfather of Eclipse strategy

Clean code definitions

Ron Jeffries [6]

In priority order, simple code:

- *Runs all the* **tests**
- *Contains* **no duplication**
- **Expresses** *all the* **design ideas** *that are in the system*
- **Minimizes** *the number of entities such as classes, methods, functions, and the like*

[6] Coinventor of the Extreme Programming (XP)

Clean code definitions

Ward Cunningham [7]

You know you are working on clean code when each routine you read turns out to be pretty much what you expected. You can call it beautiful code when **the code also makes it looks like the language was made for the problem***.*

[7] Inventor of Wiki and coinventor of eXtreme Programming

Clean code definitions

Robert C. Martin (Uncle Bob) [8]

The next time you write a line of code, remember you are an **author writing for readers** *who will judge your effort*

[8] Author of *Clean Code: A Handbook of Agile Software Craftsmanship*

Clean code definitions

- There main idea behind all definitions is the importance of the code **readability**, thinking in future software **maintenance**
- Other notions that appear in the definitions are:
  - Code that passes the tests
  - Keep the code simple and minimal
  - No code duplication
  - Code must do only one thing
- We will see some good practices for writing clean code

Clean code: Good Practices

- Use meaningful names
  - Use intention revealing names
  - Avoid *one character* names for variables
  - Avoid disinformation
  - Use searchable names
- Methods should be easy to understand
  - Should be as small as possible
  - Should do only one thing
  - Use descriptive names
  - Without *side effects*
  - Should have at most two arguments
  - Don't repeat yourself, that is, avoid duplications
- Classes should be small
  - Should have only one purpose
  - Its public interface must be minimal

Clean code: Good Practices

- Clear code with few comments is much better than complex code with lots of comments
  - Write informative comments
  - Avoid too long comments
  - Do not put useless comments
  - Avoid closing brace comments
  - Put `TODO` comments when needed
- Format your code properly
  - Indent the code
  - Leave vertical spaces for separating different *blocks of code*
  - Use braces (I.M.H.O. always)
  - Do not keep commented-out code (remember that we use version control systems)

Clean code: Code Smells

- As we have seen, there are many good practices that should be followed
- When some code do not follow the good practices we say that this code **smells**

Code smell [9]

“A code smell is a surface indication that usually corresponds to a deeper problem in the system”

- A code smell is, by definition, something that is quick to spot
- A code smell do not always indicate a problem, but it indicates that it is needed to look deeper to see if there is an underlying problem

[9] by Martin Fowler

Technical Debt

- The technical debt appears when you **postpone the refactoring** of issues in the code
  - Architecture, structure, duplication, test coverage, comments and documentation, potential bugs, complexity, code smells, coding practices, style, . . .
- It reflects the **implicit cost** of choosing a faster solution instead of using a better solution that would take longer
- The analogy with the **financial debt** term is direct: *"with borrowed money you can do something sooner than you might otherwise, but until you pay back that money you will pay interest . . . and every minute spent on not-quite-right code counts as interest on that debt"*.

Technical Debt

- Technical debt comes in the form of **extra work in the future**
  - There is no problem in borrowing against the future, as long as you pay it off
  - But technical debt must be kept always under control
- There are some aspects to be considered for prioritizing the debt:
  - Type and amount of impact
  - Duration and periodicity of the impact
  - The *age* of the debt (legacy code)
  - Its refactoring cost
  - Is it intentional or not
- *Static analysis tools* helps to detect and estimate the technical debt accumulated by the project

Static Analysis Tools

Static Analysis

Static Analysis of code is a technique to study properties of software by examining the code without actually running it

- Which properties can be studied by static analysis?
  - Duplicated code, coding standards, code coverage, code complexity, aliasing, sign, pointer, escape, liveness, nullness ...
- Some of these analysis are obtained purely through the use of rigorous mathematical methods (a.k.a. *formal methods*)
- Other static analyses are done by the application of algorithms: to search typical bug patterns or if the code complies the code standards...
- There are multiple tools that statically analyze the code and give useful information about the **quality** of the code:
  - CheckStyle, PMD, FindBugs, **SonarQube**

SonarQube

- **SonarQube**[10] is an open source platform developed for continuous inspection of software quality
- Performs automatic reviews with static analysis of code to detect *code smells*, *potential bugs*, *security vulnerabilities*, . . .
- Applies hundreds of rules on the code to measure the maintainability of source code, manage its *technical debt* and detect potential bugs

[10] `https://www.sonarqube.org/`

SonarQube Code Viewer

- **Issues** detected by SonarQube
    - **Bug**: An issue that represents something wrong in the code and must be fixed as soon as possible
    - **Code smell**: An issue that represents something wrong in the code that should be changed
    - **Vulnerability**: A security-related issue which represents a potential backdoor for attackers
- **Duplication**: Duplicated blocks of lines (more than x lines of code)
- **Coverage**: Shows the coverage of the lines of code and conditions with JUnit tests

SonarQube Quality Gate

- SonarQube includes the notion of a **quality gate**
  - Its a way to check if a project holds a minimal quality policy
  - The *gate* evaluate a set of **conditions** that a project must meet
    - If this conditions is under a threshold, the quality gate fails (the gate is closed) and the code should be fixed before continuing
  - This conditions can include the different issues: bugs and vulnerabilities, code smells, coverage or duplications
- The set of conditions and its minimal values can be configured for each project

SonarQube

- In the course we will work with our own installation of SonarQube

  `http://costa.ls.fi.upm.es:9000/sonar`

- SonarQube includes an Eclipse plugin, **SonarLint**, which includes some functionalities (not all of them), whose use in the course is widely recommended

  `https://www.sonarlint.org/eclipse/`

Further Reading

Robert C. Martin, **Clean code: A handbook of agile software craftsmanship**, 1 ed., Prentice Hall PTR, Upper Saddle River, NJ, USA, 2008.