Kristianstad University
SE-291 88 Kristianstad
Sweden
+46 44 250 30 00
www.hkr.se

Sustainable programming through good and clean code

Author: Meron Habtemichael

Author

Meron Habtemichael

Title

Sustainable programming through good and clean code

Examiner

Eric Chen

Course

Methods of sustainable programming

Programme

Software Development (Start year: 2020)

Contact

Email: meron4x@yahoo.com

Abstract

The more a software becomes complex, the more important it is to have a well-structured, good, and clean code. This report gives sight to sustainable programming through good and clean code. There are some guidelines under which developers are strongly recommended to keep track when developing or there are methods for sustainable programming. Some of these methods can be, for example, functional programming styles, writing testable code, documenting software, the philosophies or principles when creating software, uses of static code analysis tools to measure the software quality, and some more. This enables future errors to be conveniently fixed and provides understandable procedures for future developers to work on improvements and additional features. There is a saying “a code is written for a future programmer”, and it is true. That is precisely why a good, clean, and well-functional code is truly a key to a successful project.

Keywords

Good and clean code, sustainable programming, Functional programming, software principles, software testing.

Table of contents

Introduction

This report explores the true meaning behind the idea of sustainable programming through Good and Clean code. It also dives into the notions of some philosophical view of code developing approaches considered sustainable for future development and maintenance. In other words, think of achieving a Good and Clean code, what does it mean by definition? What are the methods or techniques for acquiring it? What are the advantages of having to apply it?

Along With some detailed explanations focusing on Functional programming, this report touches on the concepts of some great ideas in the subject. One of the most interesting facts lies in the fact that as technology and innovation are growing, the need for sustainable programming is increasing.

Methods

Definition of good and clean code

Diverse sources may have a variety of definitions over the idea of good and clean code, but the general understanding is similar. In this report, *Good and Clean code is defined as readable, correct, and of great performance* [8]. Well, one can say there is much to it but generally all the other definitions can be either assumed under these three words or something similar.

- *Readable*: Simple, understandable, documented, clear, structured, good naming, DRY, consistent patterns, etc.
- *Correctness*: No bugs, concise, works, solid, etc.
- *Great performance*: Efficient, Least evaluation, etc.

The following subtopics play a big role in the development of such a sustainable program through good and clean code. Let us explore them.

1. Functional programming

Functional programming is a declarative way of programming paradigm that focuses on results, not the process. It deals with what needs to be done instead

of how it is done. Functions are treated as first citizens or first-class objects. They are independent units and can be executed in any order [7]. At the heart of functional programming is thinking about the construction of software based on some fundamental defining principles [4].

1.1 Major Concepts of functional programming

It has some characteristics that make it a style. The benefits of writing code in this specific style are enormous. The following figure 1.0 demonstrates some of the main concepts of functional programming.

- Creating pure functions is one of the main practices in functional programming [6].meaning individual functions are independent on outside data (variables or functions.
- Data immutability is also a huge aspect in this area. Functional programming avoids mutating data and advising the creation of data structures to avoid changes on the existent data [4].
- The idea of Lazy evaluation is also important. It is an evaluation of an expression until its value is needed [1].
- Modularity refers to a software design technique where individual parts of a program (functions) have as little influence as possible on other parts. Each block of code as a function is a module by itself.
- Recursion means a function calling itself. This replaces the use of loops from imperative programming [1].
- A side-effect is a change in the state of the program. It can be caused by a function call or modification or changing of global variables. This approach is avoided in functional programming.
- The concept of First-class functions is that functions can be treated as objects and be stored in variables. This enables them for example to be passed to another function and returned. They are widely used in Functional programming [5].

Now the concept of functional programming is introduced and explained in this report, it is time to see the main contribution of it to the subject (Clean and Clean code).

1.2 Benefits of functional programming on clean and good code

Generally, Functional programming has many advantages that are encouraging to use. Some programming languages (JavaScript, Python, Java, and some more) have introduced features, for example, Lambdas and Streams that massively support the functional programming style. Once it comes to good and clean code, its concepts play a big role. Let us look at some of them.

- Pure functions:
  - Readability: understandable code, DRY, consistent patterns, testable.
  - Correctness: reduce the chance of creating errors, simply modifiable.
- Recursive function calls:
  - Readability: understandable, short.
  - Correctness: concise and works.
- Lazy evaluation(Using Lambdas and Streams):
  - Readability: understandable code, clean, simple.
  - Correctness: reduce the possibility of creating bugs.
  - Performance: almost double in performance, concise.
- No side-effects:
  - Readability: clean.
  - Correctness: maintainable.
- Modularity:
  - Readability: best practice, clean, understandable, structured, testable.
  - Correctness: single responsibility to reduce errors as the program runs, modifiable.
- Immutability:
  - Readability: reduce confusion.
  - Correctness: less risk of future bugs.
- First-class function:
  - Performance: reduce unnecessary evaluation in, succinct way.

Based on the notion of the above explanation, Functional programming is a good practice to obtain a good and clean code.

2. Unit testing

Unit testing is a level of testing that is simple to apply and very useful in terms of saving time and money, providing documentation, reusability, reliability, helping gauge performance, improving code coverage, reducing code complexity, and more [6]. Therefore, it is a beneficial area to be engaged in.

From Good and Clean code viewpoints, Unit testing has a significant contribution.

- Readability: reducing code complexity makes a program substantially understandable. The idea of Unit testing is reliant on separate units, so independence makes the code readable and well structured. Documentation is also an excellent tool for readability.
- Correctness: improves architecture giving rise to low bugs probability. Its overall goal is the correctness of the code when running.

3. Test-Driven Development(TDD)

It is self-explanatory, a development is driven from tests implying the tests are built first. It takes getting the hang of it and later becomes straightforward after plenty of advantages [2].

Reducing the time required for project development and code flexibility, easier maintenance, reliable solution, saving project costs, in the long run, are some of the main benefits harvested from applying TDD. “At least, it helps in planning classes and functions in a manner where each line of code is evaluated in terms of what might go wrong” [2].

The contribution of TDD to the Good and Clean code is also immense. Let us see some examples of those.

- Readability: building well-structured project and avoidance of useless or unnecessary components/features, solid and maintainable code, testable, sophisticated documentation.

- Correctness: at its core, the cycle of using the TDD approach is accurate and error avoidant [6]. This is perhaps the most useful point of TDD.
- Performance: boosts the speed of the software by prohibiting the use of non-usable features (YAGNI).

4. Software documentation

Software developers constantly find the idea of reliable documentation beneficial. It helps to keep track of all aspects of an application improving the quality of the software product. It makes it reliable due to the development, maintenance, and knowledge transfer capability it provides to other developers.

Here are some of the best practices of writing documentation.

- Include a README file containing a brief description, installation instructions, and some more about the project.
- Provide API documentation (containing some information about what a function does with its parameters or arguments and returns) and documentation of the code as well.
- Appy coding conventions such as file organization, comments, naming conventions, docstrings, etc.
- Include references, contributors, and some licensing information.

In terms of Good and Clean code: The readability and comprehension of software are significantly higher once properly documented. This results in easy-to-improve scenarios with the potential to level up the performance and accuracy of the code.

5. Development Environment

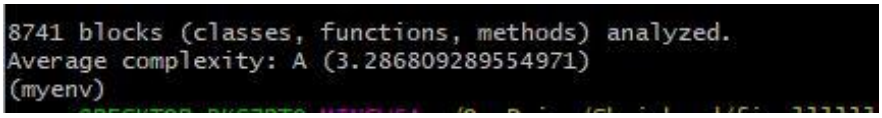
A Development Environment is mainly what developers use to create and make change without breaking anything in a live environment. Its core benefit is to allow testing of every component/feature of the code. If some bugs or errors are found, they become fixable before the project is leveraged and used in the production environment. This also contributes to the *correctness* of the code.

6. Software Philosophies

Generally, software philosophies or principles are a set of specific guidelines and recommendations that engineers or developers follow during program development. They are advised so that to attain a good and clean code that is additionally maintainable. Some of these many principles consist of KISS, POGE, YAGNI, 80 – 20 rule, POC, separation of concerns, technical debt, and many more. It is certainly a good idea to apply these principles.

7. Software quality metrics and static code analysis tools

- *Software quality metrics* focus on the quality aspects of the product, process, and the project. Refers to the size, duplication, coupling, cohesion, complexity, CRAP score, coverage, and some more about a software [3]. For example, after running Radon (a python tool that computes various matrices) on a project specified in the “previous projects” section, it was evaluated containing 8741 blocks with an average complexity of ‘A’. This indicates relatively good complexity.



- *Static code analysis* focuses on testing the code without execution. Some examples are Pylint, Flake8, Mypy, jscs, and some more. For example, after running Pylint on that project:

The overall objective of utilizing quality matrices and static code analysis is to accomplish a good and clean code using engineering measurements.

Sustainable software development

Developing software is a complex process that is performed not only in its establishment but through its continuous need for development and maintenance

to keep pace with the rapidly evolving digital world. For this reason, the necessity for sustainable software development is critical. The 6 techniques aforementioned are some of the outstanding recommendations one can apply to accomplish *sustainable development through Good and Clean code*.

Results

Previous projects

Consider the project which can be found in the link:

(https://github.com/mj6meron/dice-game-project)

Within this project, some of the upper-mentioned methods have been applied. For example:

- It is developed through Test-driven development so it can harness its benefits.
- Some of the software principles have been applied, for example: don't repeat yourself (DRY), separation of concern (making each function or a class do a specific task dealing with a specific concern), you ain't gonna use it (YAGUI – because it was developed in TDD) and a few others.
- The software is documented. All API-documentation, important files (README RELEASE and LICENCE files) are included, and the code documentation.
- Running static code analysis tools (for example Pylint, Flake8) were contributing to the cleanness and goodness of the code. It was then easier to generate automatic documentation.

Discussion

- The project mentioned above was able to benefit from the TDD approach. It contains only features to pass the test cases.
- Software principles can hugely guide a better code.
- Functional programming is also a powerful style for both the performance and appearance of a code. Its is widely used in a variety of programming

languages and many engineers recommend using it. In this report, the reason behind the focus on functional programming is because in my opinion *(the author*), it makes a code so much understandable, efficient, and correct and that is how we defined the meaning of good and clean code in this report. For example, as we discussed earlier in the functional programming section, the avoidance of loops can create great comprehensiveness and less confusion by replacing them using streams. In java 8, the use of some new features (e.g.: *records*) makes the code short and concise.

- Once a good and clean code is attained, future developers can easily maintain and develop the software.
- Thus, sustainable software development is crucial.

Summary

At the present, the tech industry is evolving so fast that it is necessary to have sustainable software development more than ever. This opens a door to some methods or techniques which can be practiced to a wide extent to have sustainable programming through good and clean code. Some practices include Functional programming, Testing, software documentation, software philosophes, quality measurement metrics, and static code analysis tools. It is highly recommended that software is built in a way that can be maintained and developed without complications, properly tested, and well documented in a good and clean way. Ultimately, success is virtually guaranteed after achieving sustainably developed software.

REFERENCES

1. Anon., n.d. *JournalDev.* [Online] Available at: https://www.journaldev.com/8693/functional-imperative-object-oriented-programming-comparison#what-are-the-characteristics-of-fp
2. Anon., n.d. *ST benefits.* [Online] Available at: https://www.exove.com/blogs/benefits-of-software-testing/#:~:text=Testing%20is%20useful%20and%20important,not%20just%20the%20QA%20personnel.
3. Anon., n.d. *Tutorialspoint.* [Online] Available at: https://www.tutorialspoint.com/software_quality_management/software_quality_management_metrics.htm
4. Anon., n.d. *What is Functional Programming? Tutorial with Example.* [Online] Available at: guru99.com/functional-programming-tutorial.html#:~:text=1%20Functional%20programming%20or%20FP%20is%20a%20way,to%20mimic%20the%20mathematical%20functions%20More%20items...
5. contributers, M., 2021. *First-Class Functions.* [Online] Available at: https://developer.mozilla.org/en-US/docs/Glossary/First-class_Function
6. Olan, M., 2003. *Testing.* [Online] Available at: https://www.researchgate.net/profile/Michael-Olan/publication/255673967_Unit_testing_Test_early_test_often/links/5581783608aea3d7096ff00c/Unit-testing-Test-early-test-often.pdf
7. Warburton, R., 2014. *Java 8 Lambdas.* s.l.:O'rReilly Media.
8. Anon., 2015. *Loup's.* [Online] Available at: https://loup-vaillant.fr/articles/good-code