# Mahantesh c sajjan

# Introduction:

Learn clean coding basics in 30 minutes

This paper in brief explains what is Clean Coding and steps to do Clean Code. Objective of this paper is to give introduction to clean coding, briefly explain some clean code principles which are easy to understand and easy to follow. It will also give brief introduction about other clean code principles for which more details could be found in internet or books.

According to me two important books that developers should not miss to read is “Refactoring” by Martin Fowler and “Clean Code” by Robert C Martin. Especially I am big fan of “Refactoring”. These two books inspired me to write this blog.

In this blog, I briefly describe about some of clean coding principles mentioned in Clean Code book and based on my experience. But I strongly recommend to go through these two books.

# DETAIL SECTION:

## WHAT IS CLEAN CODING?

Find below definition of clean coding by well-known and deeply experienced programmers

**Dave Thomas, founder of OTI, godfather of Eclipse strategy:**

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 of doing one thing. It has minimal dependencies, which are explicitly defined and provides a clear and minimal API.

**Michael Feathers, author of Working Effectively with Legacy Code:**

Clean code always looks like it was written by someone who cares

**Ron Jeffries, author of Extreme Programming Installed:**

In recent years I begin, and nearly end, with Beck’s rules of simple code. In priority order, simple code:

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

**Ward Cunningham, Inventor Wiki and Fit:**

You can call it beautiful code when the code also makes it look like the language was made for the problem

**WHY CLEAN CODING:**

We have seen that, nowadays maintenance of projects is consuming a lot budget. Making a small enhancement is taking more time as it is impacting in many places and hence more testing time is needed. New developers are not able to understand the code written by others, code is complex to understand. Clean coding will help us to solve such challenges and improve the maintainability of the project

In “Clean Code” book, it is well explained why clean code is must on existing or legacy code. Pragmatic Dave Thomas and Andy Hunt used the metaphor of broken windows to explain it.

A building with broken windows looks like nobody cares about it. So other people stop caring. They allow more windows to become broken. Eventually they actively break them. They despoil the façade with graffiti and allow garbage to collect. One broken windows stars the process toward decay.

**ATTITUDE:**

Most of developers have one common question, where is the time to follow all these principles in tight schedule of the project. Uncle Bob (Robert) explains in the “Clean Code” book as below on it. I am using the exact word from it.

This may be a bitter pill to swallow. How could this mess (**existing code smells**) be our fault (**developers**)? What about the requirements? What about the schedule? What about the stupid managers and the useless marketing types? Don’t they bear some of the blame?

No. The managers and marketers look to us for the information they need to make promises and commitments; and even they don’t look to us, we should not be shy about telling them what we think. The users look to us to validate the way the requirements will fit into the system. The project managers look to us to help work out the schedule. We (**developers**) are deeply complicit in the planning of the project and share a great deal of the responsibilities for any failures; especially if those failures have to do with bad code!

“But wait!” you say. “If I don’t do why my managers says, I’ll be fired”. Probably not. Most managers want the truth, even when they don’t act like it. Most managers want good code, even when they are obsessing about the schedule. They may defend the schedule and requirements with passion; but that’s their job. It’s your job to defend the code with equal passion.

To drive this point home, what if you were a doctor and had a patient who demanded that you stop all the silly hand-washing in preparation for surgery because it was taking too much time? Clearly the patient is boss; and yet the doctor should absolutely refuse to comply. Why? Because the doctor knows more that the patient about the risks of disease and infection. It would be unprofessional (never mind criminal) for the doctor to comply with the patient.

So too it is unprofessional for programmers to bend to the will of managers who don’t understand the risks of making messes.

Note: When hand-washing was first recommended by physicians by Ingnaz Semmelweis in 1847, it was rejected on the basis that doctors were too busy and wouldn’t have time to wash their hands between patient visits.

## CLEAN CODING PRINCIPLES:

1. **Meaningful Names**

Names are everywhere in software. We name variables, functions, arguments, classes, packages, source files etc. We name and name and name. Because we do so much of it, we need do it well.

When we want to name our child, we do think, analyses and research a lot. But when we name in software’s we do it instantly without giving a thought. Software names deserve better caring ☺

Names should reveal the intention of the Class, Functions, and Variables. If names are proper, then there is no need of comments. The content or gist of comments can be function name itself. Names should follow below principles

* Use Intentions Revealing Names
* Avoid Disinformation in Names
* Make Meaningful Distinctions
* Use Pronounceable Names
* Use Searchable Names
* Avoid Mental Mapped Names or Short Names

1. **Comments**

“Don’t comment bad code – rewrite it” by Brian W.Kernighan and P.J.Plaugher

Comments are not like Schindler’s List. Indeed, comments are, at best, a necessary evil. If our programming languages were expressive enough, or if we had the talent to subtly wield those languages to express our intent, we would not need comments very much-perhaps not at all.

I suggest to go through “Clean Code” book to understand about good comments and bad comments. Here I have mentioned in quite short.

* Comments are always failures.
* Comments lie. Not always, and not intentionally, but too often. The older a comment is, and the further away it is from the code it describes, the more likely it is to be just plain wrong. The reason is simple. Programmers can’t realistically maintain them.
* Explain yourself in the code by using right names for classes, function, variable etc. Do not worry how long the method name or class name it. But they should explain what they intent do.
* Good Comments:
  + Legal comments: Sometimes or corporate coding standards force us to write certain comments for legal reasons. For example, copyright and authorship statements are necessary and reasonable things to put into a comments at the start of each source file.
  + Informative comments: They are needed sometime to explain things like Regular expressions.
  + Explanation of Intent: Sometimes a comment goes beyond just useful information about the implementation and provides the intent behind a decision.
  + Needed for Clarification and Warning of Consequences
  + TODO comments: Some it is necessary but avoid having TODO comments as much as possible. Do the implementation of code instead of TODO comment.
  + Amplification: A comment may be used to amplify the importance of something that may otherwise seem inconsequential.

1. **Functions**

Find below the principles to be followed for functions

* Functions should be small. If a function has comment explaining a code block, then that code block can be another function and comment can be name of function.
* Functions should do one thing. They should do it well. They should do it only.
* Functions will have declaration, initializations and sieve sections in order.
* Use descriptive name for functions
* Functions should avoid more than 3 variables. If more than 3 variables, use objects
* Have no side effects. Function promises to do one thing, but it also does other hidden things. Avoid it
* Avoid OUT arguments
* Prefer Exceptions to Returning Error Codes
* DRY: Do not repeat yourself. Avoid duplication of code

1. **Classes**

* Classes should be small.
* SRP: The Single Responsibility Principle states that a class or module should have one and only one reason to change. Classes should have one responsibility – one reason to change
* Classes should have high cohesion. Classes should have a small number of instance variables. Follow the Law of Demeter
* OCP: Classes should be open for extension but closed for modifications.
* Classes should have low coupling. Dependency Inversion Principle (DIP) says, our classes should depend upon abstractions, not on concrete details.
* Classes should be easy to understand. Basically it should be like a short story reading from top to bottom. Function names should explain the story.

1. **Unit Tests**

TDD is the best way to have unit tests. TDD is a bigger topic to explain in 30 minutes☺.

1. **Formatting**

I recommend third party tools like Style cops do this job. But do not allow these tools to overtake you. If you feel, a specific rule does not make sense, then exclude it.

1. **Design and Development principles**

Clean coding could be well done by following below principles. Information about these principles could be found in internet.

* DRY : Do not Repeat Yourself
* YAGNI : You Aren’t Gonna Need It
* KISS : Keep It Simple and Stupid
* SOLID: **LSP, ISP and DIP principles are tricky.** Don’t follow it blindly. This may lead to anti patterns like Interface and Class explosion.
  + SRP: Single Responsibility Principle
  + OCP: Open Closed Principle
  + LSP: Liskov Substitution Principle
  + ISP: Interface Segregation Principle
  + DIP: Dependency Inversion Principle
* LAW OF DEMETER

## Conclusion:

In this blog I tried to explain about clean coding based on the book like “Refactoring”, “Clean Coding” and based on my experience. I strongly recommend to go through these books and understand the above mentioned design principles in depth.

Do not follow clean coding principles blindly, else some time it may become anti pattern. If you feel, it doesn’t make sense, then do not follow. But before doing that try at least once or twice. Ultimately common sense will prevail, hence don’t follow anything blindly ☺

## Questions:

1. Whether Unit testing is must of Clean Coding? Yes or No
2. Whether Clean Coding needs to be done on Legacy code? Yes or No
3. Before 1847, whether hand washing by physicians was compulsory? Yes or No
4. Whether function names need to be short or long which describe the intent of function
5. Why right names are important for classes and functions.
6. If general, is it good to have comments or not
7. Why comments are evil.
8. Give example for some good comments
9. Give example for some bad comments
10. What is the significance of DRY principle
11. Whether functions should be small or big
12. Briefly explain about Prefer Exceptions to Returning Error Codes
13. Why we should not use ÖUT”arguments
14. What is OCO and SRP
15. What is YAGNI, briefly explain about it