CLEAN CODE TO TEST CLEAN

SP Rahmesh*; Sudarshan G**,Ranjith R***

Introduction

This article is written with pragmatic approach insisting the importance of clean code and how it supports the application over a long period. Programs scripted before 1960s and 70s are still in action, taking care of major business houses in the banking, insurance, and health sector. At the same time reengineering of the applications, refactoring the code, removal of technical debts in the application, rewrite of the code and re-platforming are the major projects that being executed in the recent years, not just due to technology changes and business compulsions but also the application programs have reached a point where no further enhancements can be taken up. This paper surely will enhance the awareness within IT organizations and professionals about the importance of coding clean with a structured programming.

Key words: Clean Code, Testing, DRY Principle, Reusability, Automation

Clean Code

What is the definition for Clean Code? - According to Mac Book definition “Pleasingly graceful and stylish in appearance or manner. Pleasingly ingenious and simple. Writing code requires the disciplined use of a myriad little techniques applied through a painstakingly acquired sense of “Clean liness” .To be elegant and efficient the logic should be straightforward to make it hard for bugs to hide. The maintenance can be made easy and error handling be complete to ensure the entire program is intact. Performance of the code must be managed at optimal level so as not to tempt programmer to make the code messy.

The Code-Sense is the Key for programmers. Some of us are born with it otherwise have to fight to acquire it. Not only does it let us see whether code is good or bad, but also shows us a strategy for applying our discipline to transform bad code into clean code.

A programmer without ‘**Code-Sense**’ can look at a messy module and recognize the mess but will have no idea what to do about it. At the same time with a clear code-sense a programmer will see all options and possibilities to improve the code. This helps the programmer to identify

*Dr. SP Rahmesh – Director, Thryve Digital Health LLP; sulur.rahmesh@thryvedigital.com
**Sudharshan G – Adviser, ***Ranjith R Sr. Architect - Thryve Digital Health LLP;

the best variation and guide to plot a plan to transform the code to make it efficient.In short, the programmer who writes clean code is an artist who can take a blank screen through a series of transformation until it is an elegantly coded system.

Primal Conundrum

Messy code slows you down instantly and forces you to miss deadline. Only way to make the deadline is to keep the code as clean as possible at all times. Clean code means a code is readable, simple and orderly. An Architect should pay appropriate attention to maintain it as clean as possible.

'First Make the code work then make it Right.'

The Boy Scout Rule

It's not enough to write the code well but the code has to be kept clean overtime. Older codes get rotten and degrade as time passes, but every effort should be taken to prevent the degradation.

"Leave the campground Cleaner than you found it"

Few TIPS are given below as guidelines for programmers to adopt to get a good outcome.

1. Program comments should be appropriate explaining technical / design / domain aspects and NEVER clutter up
2. Obsolete comments get older and forgotten quickly, so it is best NOT to write comments that will become obsolete soon. IT's good to write only relevant information properly.
3. Avoid redundant comments and repeat statements.
4. Use right words, punctuation with correct grammar and don't ramble.
5. Commented out Code – code sits there,rots, and becomes less relevant day by day. Further it pollutes the module, distracts the programmer who reads it and it's like abomination. When a programmer looks at unwanted commented code, it's appropriate to delete it. Anyway, the source of control and version history remains in the recovery process to look back if need arises.
6. Don't suffer for the commented-out code to survive.

7. Keep It Simple: Meaning, ensure the code is written as readable as possible without adding complexity to it and over complicate. Write the code by adopting the industry's best practices and structured programming techniques. Ensure the CPU is not overloaded unnecessarily by making the code like Spaghetti.
8. Write the code in such a way that you are able to understand it at any time later, as the code will need to undergo modifications to cater the business needs. Every complex Algorithms and business Logics in the program should be written with appropriate comments for future use.
9. Adopt the DRY Principle while coding – [**D**on't **R**epeat **Y**ourself] – DRY principle formulated by Hunt & Dave Thamas in a pragmatic way. Don't retype the code which already exists within the application, instead Reuse the code in the form of a function or stored procedure, so that the application gains efficiency and increases the productivity without any specific effort. Use Framework features instead of any custom code to enhance the ability of the application to serve the customers better.
10. Adhere to coding standards and "Never Deviate."

What a Programmer should do?

It's unprofessional for programmers to bend to the will of the Managers who don't understand the risks of making the code MESS most of the times. Programmers to meet the deadline can't make the Code Mess as it will slow down the productivity. The only way to make the deadline is the keep the code as clean as possible at all times as you can.

"MESS will force you to MISS the Deadline."

As the Mess always lowers the productivity, the Programmer should stay vigilant even though, horrific pressure is put on the coders to show incremental productivity. When too many programmers get added to meet the deadline adds more mess asymptomatically. The programmers should not shy away to tell the truth but convey politely instead of violently.

What are the significant focus areas and expectations?

Code Maintainability: Never hard code any constants in the code but it should be configured and externalized for ease of supportability.

Readability: The code should be self-explanatory [get a feel of story reading while going through the code].

Testability: Code should be easy to test as testing with interfacing applications is critical aspect in testing. It's hard to Mock the interface testing always.

Debuggability: Program should be easy to debug to fix issues and to find root causes faster.

Reusability: Consider reusable services, functions, components [DRY Principle]

Extendibility: The code should be flexible enough to extend and modify to add any business logics and needs.

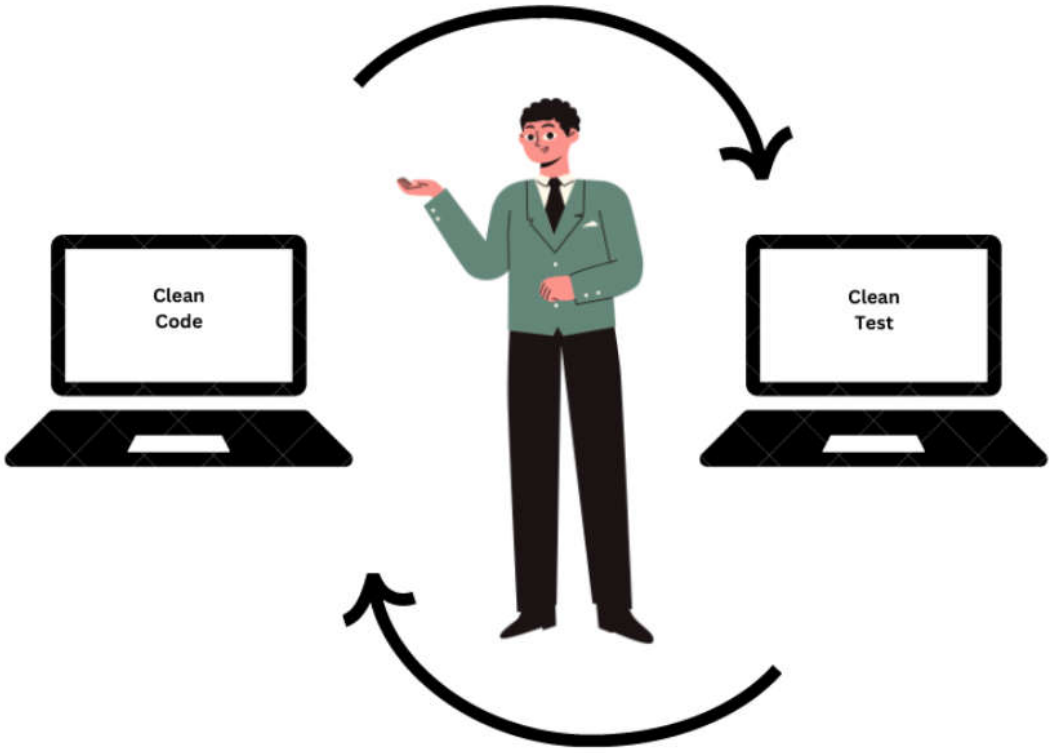
HAPPY CODING

Code Metrics

Software Metrics relating to source code that are widely adopted in the industry are Size measure and Complexity. Unfamiliar measures are code coupling, cohesion, and inheritance. It's a good practice to measure these five metrics for better code maintenance.

1. Size measure refers to LOC, Function Point, Feature Point and etc.
2. Complexity of source code could affect code modularity and maintainability.
3. Coupling refers to number of connections a file or class has with other files or components. The assumption is that when the lower the coupling is better for code modularity.
4. Cohesion measures how strong the responsibilities within a code unit are related. The rationale behind is the belief that code units [files / classes / procedures] should focus on single functionality which can improve the code maintainability.
5. Inheritance based metrics relates to object-oriented code only. When the inheritance hierarchies are less complex, then it's easy to understand and maintain.

Good hart's Law and Effect of Measuring is '**When a measure becomes a target, it ceases to be a good measure.**'


Clean
Code
Clean
Test

Test Clean

Why software testing is important?

Hardly anyone can object to the necessity of quality controls in the development of software. Late deliveries or software defects can damage a brand's reputation and lead to frustration and end up losing the customers. In extreme cases, a bug or defect can affect interrelated systems or cause serious malfunctions. Consider Nissan's recall of more than 1 million cars because of a software flaw in the airbag sensor detectors or a software bug that derailed the launch of a $1.2 billion military satellite. Software errors in the U.S. cost the economy $1.1 trillion in assets in 2016. They also affected 4.4 billion customers. These numbers speak for them selves and emphasizes the magnitude of testing.

Although testing itself costs money, companies can save millions annually in development and support if they have good testing technology and quality assurance processes in place. Early software testing uncovers problems before a product hits the market. The earlier development teams receive testing feedback, the sooner they can address issues such as:

- Architectural weaknesses
- Poor design decisions
- Invalid or incorrect functionality
- Security vulnerabilities
- Scalability issues

What is clean testing?

A clean test also known as pure test, is a test designed to have a single clear objective and to be free from any side effects or dependencies on other parts of the system. Clean tests are important in software development because they are more reliable, easier to maintain and provide faster feedback on the quality of the code being tested.

Here are some characteristics of a clean test:

Isolated: A clean test should be isolated from other parts of the system and should not have any dependencies on external resources such as databases or web services. This ensures that the test is not affected by changes in other parts of the system and that it can be run independently.

Repeatable: A clean test should be repeatable and produce the same result each time it is run. This allows developers to easily reproduce and fix any issues found during testing.

Self-contained: A clean test should be self-contained and not rely on any external state or conditions. This ensures that the test can be run in any environment and that it does not produce unexpected results due to external factors.

Clear and concise: A clean test should be clear, concise with a single objective. This makes it easier for developers to understand and modify the test and to identify to fix any issues found during testing.

An automated test is not always good…

"Automate what you can, but always remember the human touch."

Automated testing is a double-edged sword which has recompenses such as improved efficiency and accuracy, but it also has some shortcomings that should be accounted.

1. **Inceptive Cost**: Automated testing tools and frameworks can be astronomically expensive to set up and maintain, especially for organizations with limited budgets.

2. **Complexity**: Automated testing can be complex to set up and maintain, especially for large and complex systems. This requires specialized skills and knowledge, which can be a challenge to find and retain.

3. **Maintenance Overhead**: Automated tests need to be updated and maintained whenever the system upgrades, which can be time-consuming and resource intensive.

4. **False Positives and Negatives**: Automated tests may produce false results, such as false positives (when a test fails even though the system is functioning correctly) or false negatives (when a test passes even though there is a problem with the system).

5.**Limited Test Coverage**: Automated tests may not cover every possible scenario or edge case, leading to gaps in testing coverage.

6. **Dependence on Automated Tests**: Over-reliance on automated tests can lead to complacency and a reduction in manual testing, which can result in missed bugs and vulnerabilities.

7. **Inflexibility**: Automated tests may not be able to adapt to changing requirements or new technologies, leading to outdated tests that no longer provide value.

Tips and Techniques:

Understand the requirements: It is essential for testers to understand the requirements and the expected behavior of the software being tested. This helps to ensure that the testing is aligned with the goals of the project and that the software meets the expectations of the stakeholders.

Use a variety of testing techniques: Testers should use a variety of testing techniques such as functional, performance, security, and usability testing to ensure that the software is thoroughly validated, all issues are identified and addressed.

Create a comprehensive test plan: A comprehensive test plan outlines the testing approach, the testing scope, and the test cases. It helps testers to stay organized, focused and ensures that all aspects of the software are tested.

Collaborate with the development team: Testers should work closely with the development team to understand the code and to identify potential issues early in the development cycle. This helps to reduce the time and effort required for testing and improves the overall quality of the software.

Use test automation: Test automation can help to speed up testing, reduce human error and increase test coverage. Testers should use test automation tools to automate repetitive tasks and perform regression testing.

Report issues effectively: Testers should report issues clearly and accurately, including detailed steps to reproduce the issue and any relevant screenshots or logs. This helps developers to quickly identify and fix issues.

Continuously improve: Testers should continuously evaluate their testing processes, techniques and look for ways to improve them. This helps to increase efficiency, reduce costs and improve the quality of the software being tested.

By following these tips, testers can help to ensure that the software is thoroughly tested, all issues are identified, addressed and the quality of the software gets enhanced.

Ad hoc testing:

Tester's Intuition: Usually Planned testing enables catching a certain type of defects. Though planned tests help in boosting the tester’s confidence, it is the tester’s intuition that often finds critical defects. This is because none of the specifications can be considered complete to provide all the perspectives that are needed for testing. These perspectives evolve dynamically when testers

think "out of the box". Hence a tester's intuition plays a significant role. Ad hoc testing is done to explore the undiscovered areas in the product by using intuition, previous experience in working with the product, expert knowledge of the platform or technology, and experience of testing a similar product. Ad hoc testing uncovers defects that are not covered by planned testing.

Use of contemporary practices in testing:

Traditionally testing artifacts were usually prepared as word documents which needs a significant manual effort of the tester. With the advancement of the tools, manual preparation of test artifacts or evidence can be avoided, and they can be prepared as recorded video for instance. Such improvements can reduce the time spent by the tester and also can avoid human errors.

Exploratory testing is an approach where testers explore the software without a pre-defined test plan or script. This allows for more flexibility and creativity in finding defects and identifying areas for improvement. Risk based testing approach that focuses on testing the most critical and high-risk areas of the software first. This helps to ensure that the most important functionality is thoroughly tested and reduces the risk of critical defects being missed.

Reduce & Reuse

Reduce:

Most of the products have a predefined automation suit, but running the entire suit may not be right for every requirement as certain change may be very small and it may not require the entire suit to run. Running the entire suit could be resource-consuming which in turn will raise the cost and lot of time will also be required. To avoid this issue, every test case can be tagged to components such as code blocks, database tables etc. An automated system or method could be built to select the test case from the test library.

Reuse:

In software testing, the concept of reuse refers to the practice of using previously developed and tested components, such as test cases, test scripts, test plans, and test data, in new or modified software applications. Reusing these components can save time and effort, reduce the risk of errors, and increase the efficiency and effectiveness of testing.

Here are some ways to apply the concept of reuse in testing:

Test case reuse: If the previously developed test cases that are applicable to a new project or application, one can reuse them instead of starting from scratch. This can save you time and effort and help ensure that the new application is thoroughly tested.

Test script reuse: If the automated test scripts developed already suits the requirement, then that can be reused in any new project, which could save time and reduce the risk of errors by using them instead of writing new scripts.

Test data reuse: If the previously used test data for an application is saved in repository, which can be reused to test a new application that has similar requirements. This will save time and effort and help to ensure that the new application is thoroughly tested.

Test plan reuse: It's always a good habit to develop reusable test plans, which can be reused instead of starting from scratch. This would result in huge effort savings and cost as well.

Test framework reuse: Any development of a test framework has to be built with reuse features in it, so that testing productivity can be systematically enhanced.

Conclusion:

In today's software development industry, the importance of clean code and clean tests cannot be overstated. By adopting the principles of clean code and clean tests, developers can produce software that is more maintainable, extensible, and resilient. They help ensure that the codebase is reliable, functional, and meets the requirements of the end-users. Furthermore, clean code and clean tests help facilitate collaboration between team members, as they provide a common understanding of the codebase and ensure that everyone is working towards the same goal. In conclusion, the adoption of clean code and clean tests is essential to building high-quality software that meets the needs of end-users for long years. Overall, the concept of reuse in Coding and testing can help improve the efficiency and effectiveness of the entire SDLC Processes, while also reducing the risk of errors and ensuring that applications are thoroughly verified for customer's use.

References

1. "Software Testing: Principles and Practices" by Srinivasan Desikan and Gopalaswamy Ramesh, 2006
2. A Decision-Support System Approach to Economics-Driven Modularity Evaluation by Yuanfang Cai and Hong-Mei Chen, in Economics-Driven Software Architecture, 2014
3. Clean Code: A Handbook of Agile Software Craftsmanship by Robert C. Martin, 2008
4. CN103412818 - AUTOMATION TESTING METHOD AND SYSTEM patent.
5. https://www.ibm.com/in-en/topics/software-testing
6. https://uilicious.com/ blog/pros-cons-automated-testing
7. xUnit Test Patterns: Refactoring Test Code