Open and Closed Software Development Models :A Study and Application in Life

Mr. Ashok Raj R[1], Dr. S Dinakaran[2], Dr. P Saravanan[3]

[1]Associate Professor, Department of Computer Applications, T. John College
[2]Associate Professor and HoD, Department of Computer Applications, T. John College
[3]Assistant Professor, Department of Computer Applications, T. John College

Abstract:
The design methods and software development lifecycle methodologies apply to areas outside the software environment. As a Technologies & IT head and Professor, I learn, implement and teach many software principles in day-to-day life both at the office and home. After knowing the power of Agile outside the software development environment, I often self-retrospect about how well I can make use of the computer science knowledge I am acquiring in my life. Hence for my analysis, I attempt to study different SDLC (Software Development Life-Cycle) processes that might be usable in life. I assume if the model such as Agile is a useful tool, then the others can also have room for application in my life. The software types I will study is open source. My work verifies open and closed software standards. To make my work brief and focused, I will focus primarily on Linux and Agile for Open Source (OS) and Closed Software (CS) models, respectively. Both open and closed software standards extensively impact on the software revolution. Especially closed software standards have famous beyond the software engineering environment. Before the research, I thought first to describe the commonalities that they have. During my study, I was able to identify the positive attributes of methods that define the open-source development model, which can help me to do productive things in my life. Software engineering standards, processes, guidelines, methods and tools and can be used as a catalyst to improve our daily lives.

Keywords: Open-Source (OS), Closed-Software (CS), Software Development Life-Cycle (SDLC)

1. INTRODUCTION

The description of the development process of the Linux kernel by Robert Love contradicts my findings. Love describes the process of open-source development as "chaos." I would suggest that the process of open-source development - highly well organized and strictly controlled.

One of my study's aims is to explain the underlying paradigms, methodologies, and practices describing OS and CS processes to the reader. The study would compare the critical activities of CS and OS to assess the reader's similarities and differences. I will concentrate on results on which methodologies lead to better performance of software. My findings shall link the reader uses development models and their commercial applications in their life. I will quote personal experience and other relevant examples.

A. Target readers and Plan

The target audience of this paper is the one who is interested and willing to learn and understanding on how software development paradigms work and want to know how those processes are incorporated in other own lives. The goal is to explore and equate few models in software development, and illustrate through examples and case studies that these can bring you a better parent, wife, manager, engineer, physician, sportsperson or the role that you dream of being, by applying the processes used in those models.

Firstly, I will introduce the required basic definitions and steps used in few models to the audience concerning the context. I assure you that only preliminary of computer science concepts I will explore which are suitable for the understanding the topics, and I recommend the audience to refer additional materials based on their needs of interest. Basic of overall understanding and a wish to think differently are required if you want to tailor your life using programming paradigms.

Finally, I will brief some of the research examples to map source code healthiness and security in software developed using OS standards. Then will look into comparison properties and methods of OS and CS models and touch upon a few tools and design patterns used in OS development. I will also attempt to explain my personal life experiences as case studies to explain that CS and OS possess that the audience can attempt to implement.

B. Contradiction

Per the statements of the Cathedral and Bazaar OS models are the innovation of the industry and states that conventional software inhibits maturity and improvement [1]. This truth induced my eagerness to understand more about OS software development processes. In contrast, OS design processes are well layered and explained. It encourages collaboration and learning from other sources, users, and listening to people. The guidelines, standards, and processes ensure code quality and integrity. The above are well detailed and make sure that the users are more systematic in their understanding of implementing programs. The practices which define OS have shown that it is the "chaos" that Robert Love states. The significance of how vital the OS is in our world is understandable by our government's initiatives taken by establishing ICFOSS (International Centre for Free and Open-Source Software), which is an autonomous organization set up by the Government of Kerala to support OS. It gives thrust me to learn various aspects of the OS development processes.

2. DEFINITIONS

A. Background

In CS software development, the author or owner or the developer does not release the source code as part of the release to the users; they maintain the secrecy of the source code. The owners release the binaries of executable applications or libraries to the user. They can also patent and own the intellectual property rights and hide the source from others. Many software suppliers follow this process, and they claim it as a trade secret which includes Microsoft's Windows OS. Few others to name are Internet Explorer, MS Office and Visual Studio.

Whereas OS community developers allow the users to the application or library source code, and it is open to the entire world. This process allows contributions from many sources to strengthen the design and implementation of the OS. OS can be distributed with different licensing schemes. The ultimate difference between OS and CS is code availability. Race, region, secrecy, geography, sex, copyrights do not apply to the development of OS. Contributing in OS application development is voluntary [4]. We shall compare the benefits and methods of both the models — for example the OS browser Mozilla and OS operating system Linux.

B. OS (Linux)

OS development is assumed as "a great babbling bazaar of differing agendas and approaches out of which a coherent and stable system could seemingly emerge only by a succession of miracles [1]."

We shall consider the OS operating system Linux which was developed by a student at the University of Helsinki in 1991. Linus Torvalds envisaged that the source should be modifiable by anybody to satisfy his needs [2]. Such projects were not existing at that point of time hence he started a new project. Linux started grew rapidly due to valuable contributions and today numerous applications implements Linux kernel which can run on many devices that we obviously use daily.

C. Design

Linux development - done in several releases where developers implement new functionalities until they release the functionality [2]. This process has evident development stages as per the needs of the customer. The different stages are design, review, peer review and integration into the kernel. The design phase includes documentation of requirement and will explain the implementation details also.

The preliminary documentation includes the users and problem statement; it may also include legal, regulatory, and other information if needed to define the scope of the project context. The design stage is a vital part in the development process, and the subsequent review will give insight into the action plan and time estimates for implementation. If implemented without deviation, the documentation optimizes the time and money in the development process. Usually, it is done by a group of people with a common interest.

D. Review

The early review posts the design documentation to a group of people by email, and during the first review, a community of peer developers subscribe for email for specific features. In this process, the feedback received about the initial design validated to check whether the initial design solves the original problem or not. The comments about the peer developers during the first review helps in freezing the design. The documentation is lacking clarity in meeting the project's goal; and refined multiple times until the specification meets projects' needs.

Apparently, in the next detailed analysis phase, the Linux kernel is managed by subsystem teams responsible for managing branches of the project. A rigorous design & implementation study is done by the Linux kernel maintainers at this stage. Later the formal review and extensive community scrutiny is done while merging the feature with beeline. The process is continuous, with several debugging and testing cycles. Once the community and maintainers approve the implementation, and testing proves no regression bugs during integration, stable release of the patch happens [5]. The graphical representation of the development phases shown in Figure 1.

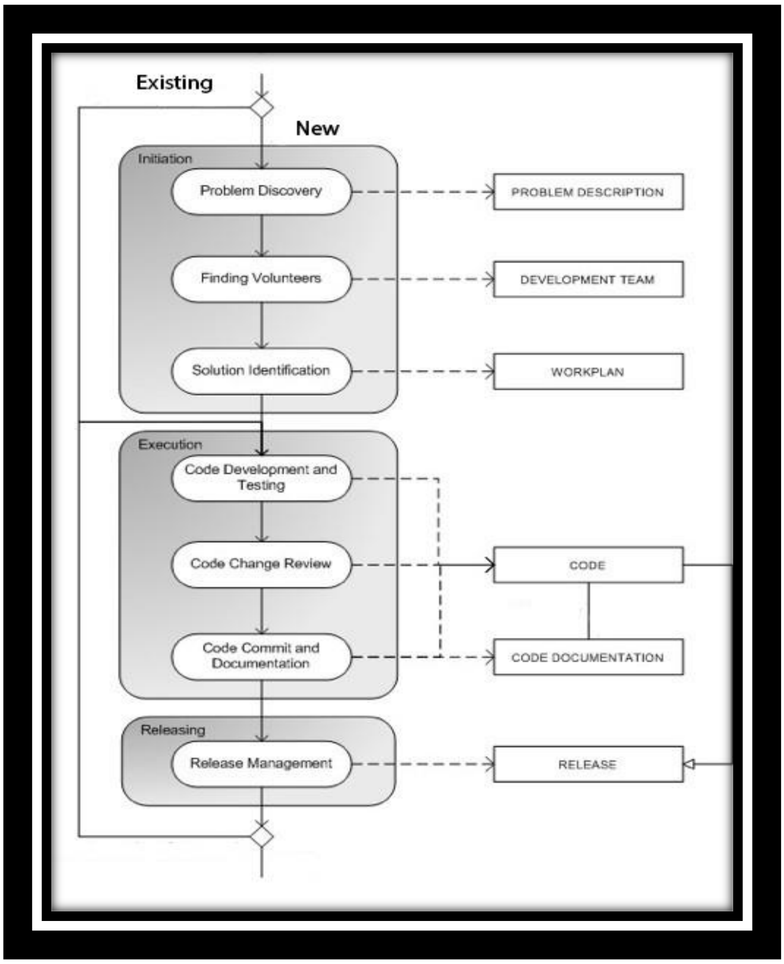


Fig. 1. Example of OS development process

E. CS – Agile model

CS development may follow the model, such as Agile, which uses necessary activities such as communication, modification, and perception. Agile uses methodologies for iterative process recommended in software engineering. Per Agile method works on the belief that teamwork, synergy, and collaboration work well. It gives importance to seamless integration, planning, and testing.

The Agile starts with requirement documentation and needs different stakeholders involve in writing use cases. The end-users should determine the project requirements. Each iteration requires user case writing, task adjustment, team planning, design, pair programming, and testing. In Every iteration, the working software is made that is tested by customer to provide feedback frequently. Frequent behavioral adjustment required on a need basis, in order to become efficient in collaborative Agile properties [11]. While I learned Agile methodologies and started practicing it, I was able to connect principles to problems that I attempt to solve at work. The core strength of Agile is evident once understood in software engineering. Collaborative work, reviews, testing, constant & regular feedback, understanding new issues, tracking schedule, and giving power to team members are good practices.

Figure 2 depicts the Agile development model to help the audience to visualize the iterative nature of Agile model.

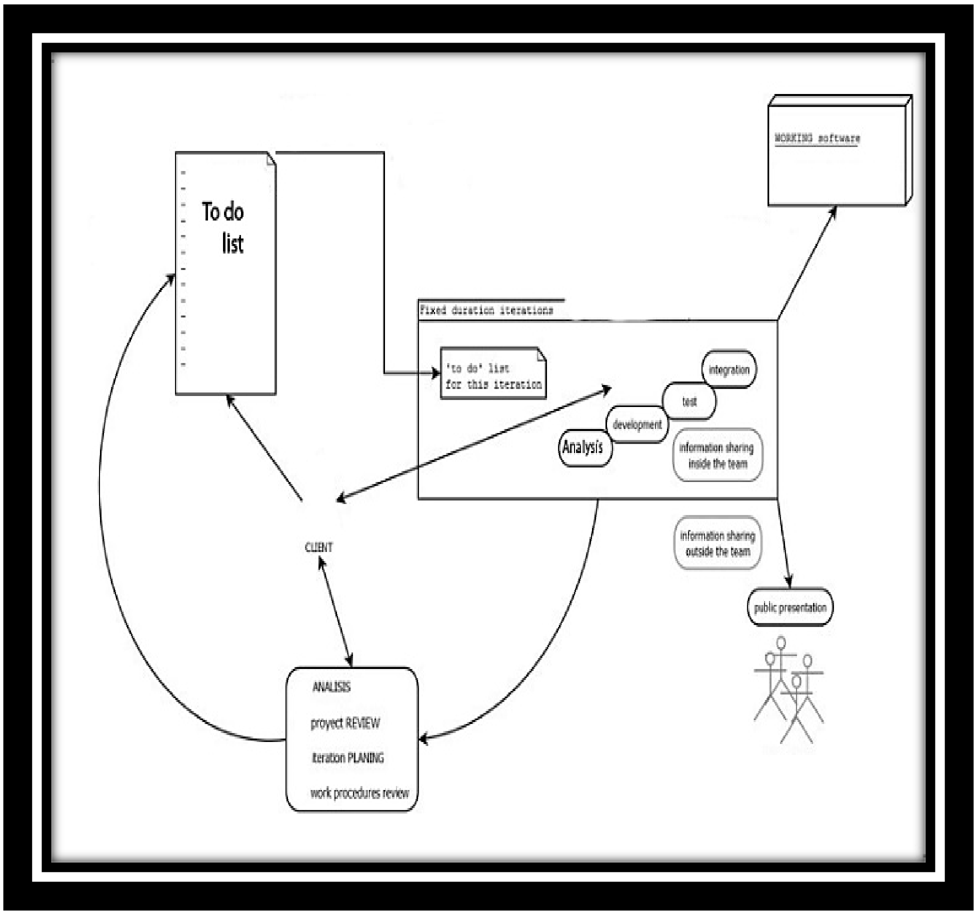


Fig. 2. Example of Agile Methodology

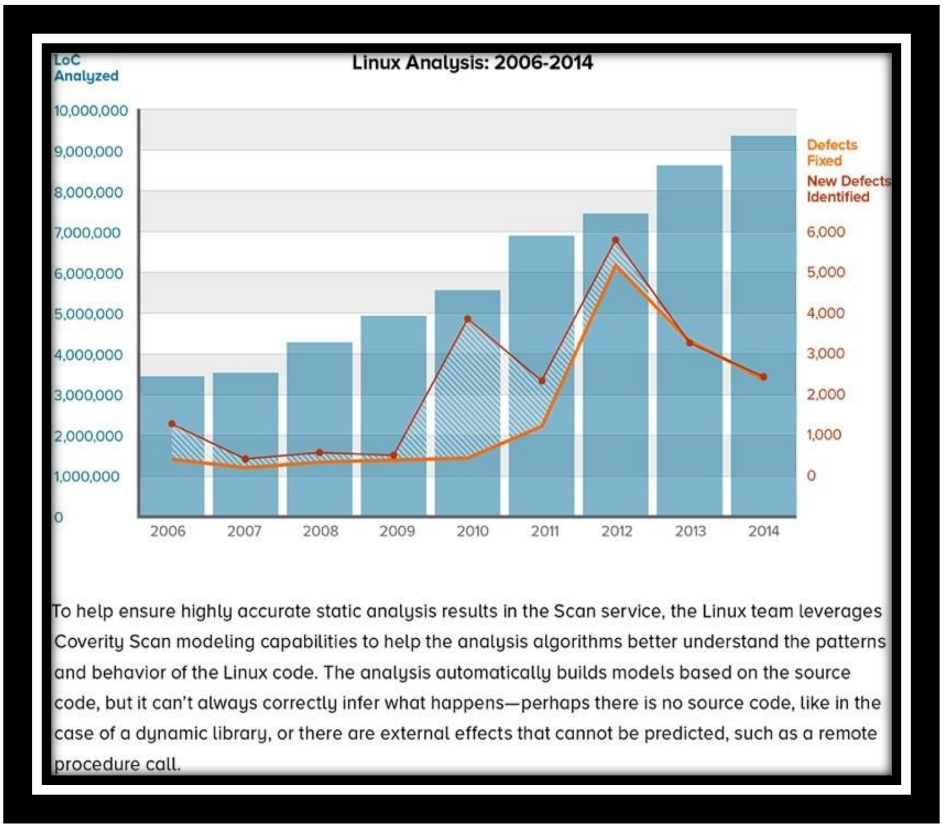
3. RESEARCH

The 2015 Future of OS Survey is an analysis conducted by Black Duck. The study resulted in the understanding that the companies using OS is continuously increasing [7] and 78 per cent of businesses use OS to run their business.

A. Issues

To understand why 78 per cent of companies use OS, I analyzed the studies. Also, I did some study on which paradigm produces the best code quality and security. The analysis of 4 operating systems FreeBSD, Linux, Solaris, and the Windows kernels for its' code quality using metrics comparing OS & CS found no difference [8]. However, further analysis and conclusions made Coverity that has verified ten billion source lines of OS and CS found different results. US Department of Homeland security runs the company Coverity [10]. Coverity Scan (currently managed by Synopsis), is the world's largest research project concentrating on OS quality and safety [9]. Synopsis does a free scan and source analysis to the OS and CS stakeholders. The Coverity and Synopsis tools provide developers code quality and security vulnerability reports, and they are industry-standard service providers [9].
The Coverity reports that from 2006 through 2015, it found that “Linux (OSS) remains the benchmark for quality” [10] and concludes that OS has a lower defect density compared to CS. Raymond theorizes that “given enough eyeballs, all bugs are shallow [1].". In The Cathedral and the Bazaar, Eric Raymond postulated fifteen years before for this consistent outcome. The effort put in the depth and breadth of

review makes reliable code. The feedback cycle that the OS has relevance to defect density analysis. After Raymond's hypothesis, Linux focuses and maintains this myth. Collaborative & exponential efforts put on review increases the code coverage, and additional testing with security audits uncovers remaining bugs that the reviewer's miss. Figure 3 shows defects analysis of the Coverity scan.



(Coverity and Synopsis Scan [10])

Fig. 3. Linux Defect Analysis

B. Security

A critical & crazy finding in 2014 Coverity study conveys that although OS results in lesser defect density and good code quality, CS is more manageable and secure [9]. Coverity states that OS is becoming more familiar and has more features. People need software to do certain things like OS, instead of bug fixing during the development. Whereas, in CS, security takes the highest precedence [9]. Analyzing results over eight years, the Coverity Scan suggests that both OS and CS are improving. Case studies on Linux observed that while seeing new bugs, they were easier to fix. When the code gets aged, the errors are harder to fix [9]. Figure 3 shows data representation for the reader.

Research shows that it is not clearly evident why OS is preferred over CS. The code quality and development speed are essential, but, compliance and security are highly substantial - according to my experience. This logical difference explains findings in the Black Duck survey analysis on how companies choose OS manage their OS features.

The survey indicates that above 55 per cent of enterprises who use OS said that they lack a formal policy for OS use [7]. The survey also found that below 16 per cent of the enterprise using OS use automation testing method [7]. About 50 per cent of the companies using OS are not satisfied concerning security vulnerabilities and even less plan to monitor OS for cybersecurity. The results show that such enterprises need to have formal and documented policies and guidelines, but unknowingly, choose development speed and implementation of new functionalities over security and compliance.

Ideally, both OS and CS have pros and cons. However, a method which uses advantages of both methods will provide the best results. It should provide harmony of security and speed. The combination of values and stages in the OS model, the tools and compliance standards of CS in adherence to software engineering principles will ensure quality code and security.

4. COMPARING PARADIGMS AND PRINCIPLES

I analyze a few principles in Agile Manifesto [11] and OS best practices to compare both models. Both models depict some similarity in many ways and share some common core properties. For example, CS and OS models share the property that delivering working code for feedback frequently. The practice of all software engineering is "release early and often". Delivering working software illustrates optimal changes due to regular feedback. For design comparisons, we need to discuss the processes more in-depth.

A. Role of the Customer

In the Agile process, the stakeholders play vital role in requirements, user stories and in designing use cases. User stories identify the type and role of users, what and why they want it. The Linux Foundation provides questioner for the developer to ask to layout the initial plan. During the planning of a kernel development project, the developer shall ask question; “What, exactly, is the problem that needs to be solved? Who are the users? Which use-cases need a solution? How does the kernel lack in addressing currently?” [5] Both methodologies meticulously attempt to solve the issues from the users' perspective. The agile process supports a change in requirements throughout the iterative process to evolve over the project life-cycle. Initial requirements may be very ambiguous, but continuous feedback refines the initial requirements and brings clarity [11]. The OS development starts with precise user requirements, but in an Agile process, new functionalities and requirement development needed continuously. In contrast to other models in OS, the user can be the developer, and the community members decide on feature implementation.

B. Motivation and Teamwork

Motivation is an essential parameter in both OS and CS development process. Implementing software is teamwork where motivation is a mandatory parameter in the success. If the right kind of motivation and given empowerment, the developers will get the job done [11]. Right kind of motivation is a public property of both OS & CS principles. OS is motivation-driven, and developers can choose a project to work which motivates them. Because OSS is voluntary, the developer need not choose a project which lacks motivation, so interest in the project is assured. Teamwork and peer cooperation are vital to the design process of OS, also demonstrates the similarity of this core principle.

One another vital aspect of the Agile method is face-to-face discussions with stakeholders [11]. The OS model varies in this aspect, where limited direct communication happens because the stakeholders, scattered across the world. However, there is a significant amount of written communication that happens. Linux and other OS community have volumes of communication and review over different community tools such as email groups. Though face-to-face communication lacks in OS, the peer involvement is same across paradigms.

In the Agile process, the frequency of builds measures the speed of the project. Also, the simple design detail and analytical design procedures make projects agile [11]. However, both OS and CS have this

property. "Make sure that the kernel runs perfectly at all times on all machines, which you can lay your hands on [5]." - Says Andrew Morton. Both OS and CS reliability is essential. Furthermore, in both the project tracking is done using board and burn-down in order to maintain a constant speed.
Overall, the Agile process encourages that the team represent together and change their actions in order to be more efficient [11]. OS communities do self-criticize and stick to the schedule well. Linux and Apache both started starting the organizational structure and devised to adopt foundations employees, maintainers, management, and executives [12].

5. TESTING AIDS AND SAFETY

A. Testing

In Software Engineering, we need to use, the refactoring and usability analysis standards to assess the code & UI sanity levels. The term "smell" refers to flaws in the design or code which might end in bugs. Both CS and OS use several tools in each iteration to identify the smells during development. The testing tools test the source code. For example, Linux uses a “lockdep” that measures dependencies among different states [13], and for Linux kernel development “poisons” - a debugger is used to free the memory which could prevent Segmentation faults and security [14]. The criteria for best practices during free OS development (FLOSS) has created a guidelines document for quality assurance. The bug tracking systems, such as ClearQuest used to track the issues and changes by OS developers. Also, Linux requires the circulation of issue status through email list. Also, OS developers the source code coverage needs to be 100% using automated testing tools [15]. FLOSS also requires continuous testing using code analysis tool.

B. Security

An OS project should have a lead programmer who knows about developing secure software. OS should implement user access permission. Based on the privilege granted the access to the resources is ensured. The programs and users of the program should work using the of authority rights to perform tasks. OS must have safety procedures to grant access to shared resources by the system users. The design must implement a process that requires two keys to access reserved resources. The method must be open and not be secret. The last principle is psychological acceptability, in which the interface design implementation is simple so that users can apply them correctly [16]. OS uses an identity system like badging to encourage adherence to OS security, design and coding practices [15].

6. DESIGN PATTERNS

The use of design patterns is vital in software engineering. A design pattern is a known solution to a known problem [17]. Both CS and OS utilize design patterns as part of the design.
The kernel programmers require to engage in peer review while doing the design by posting the plan to the relevant people. Reviews will help to identify deviations and mistakes, and this early review may help in avoiding rework and optimize the schedule of the project. Posting email to a distribution list is the process in OS development to uncover errors early. “Code which reinvents existing wheels is not only wasteful; it will also not be accepted into the mainline kernel [5]." OS members are key to current and evolving design patterns.
The Model-View-Controller (MVC) Pattern uses a proven design for implementing UI. The complete discussion of design patterns is outside the scope, but it is important to the context of this paper, to

mention design patterns do exist in OS. An example of a design pattern used in OS (Linux kernel) process is using a counting variable to manage the system resources.
Reference counting tracks the resources which are free and not free so that kernel manages memory [13]. The Linux kernel uses simple and smaller structures. An example of this how kernel handles the task structures for the processes. It uses embedded head nodes to construct a linked list, instead linked list structures.

7. LIFE

A. Agile for a Child education

The values of Software Engineering methodologies hold in common can be tuned and applied in many areas. Bruce Feiler's case study used Agile to manage his family [3]. The casual meetings during tea and dinner with Family increased communication, and everyone was able to suggest input to overcome problems. The approach helped to track family challenges, and Feiler gave his children to recommend punishments. Using just a few Agile processes, Feiler was able to change his team, and the key is creating a culture of adaptation. When goals, divided into small steps, achievements are simple, and frequent tracking is possible. The continuous retrospection and feedback from others help to stay on course to the goal [3].

B. Own Goals

Like him, I implement many of these methodologies to help and reach my personal/official goals. For instance, iterative feedback from my clients and colleagues helps myself in solving overcoming my problems. The process of self-retrospection and criticism from peers brings virtual pathways leading to constructive ideas and solutions that I face. The smaller- primitive tasks approach helps them to achieve it smoothly. We use the term primitive to refer to a problem that we cannot break down. With the core software engineering process in my mind, I teach the importance of correctness and completeness along with simplicity during employee training. My employees and peers, equipped to be creative and productive and change themselves regularly. I follow these methods every day of my life.

C. Challenges

Start each day with a meeting after prayer in the morning, and this will allow your team or family to work towards goals. Always ask constructive feedback from fellow-workers, family, or whoever. Reduce the problems into small primitive tasks, and this will help you find the solution complex problems faster and efficiently. Always think about alternate ways to address challenges; this is done by self-evaluation of yourself, not as a team. These ideas, many others in software engineering, are tools we can use to excel in our lives every day.

8. CONCLUSION

Both CS and OS models forecast many of the common core values. Both emphasizes on frequent build of working software, reviews, and verification-validation. The concepts encourage easy to approach and simple design — more stress, given for the culture of collaborative teamwork. Patience in listening & understanding the users during design and development can bring better clarity and innovation. Equally, it is so important to consider peer and self-reflection, an open mind towards change, and they concentrate on doing the correct right, even in the absence of others.
The similarity of these models was surprising to learn. When I started this work, the idea that CS and OS methodologies shared common values was a contradiction. I thought OS for developing complex

systems with large distributed teams writing code in chaotic and unorganized nature. I was surprised to find that the OS development process is so structured and well defined.

There is huge to learn and follow from OS development methods. My hope for the readers is to be at least attempt to implement some of the processes from a software engineering in your life. I learned like Agile, and I can utilize OS models and methodologies to design patterns to follow in life. I, look forward to studying more methods, processes, guidelines, tools and templates to see how humans can use them outside the software industry.

REFERENCES

1. E. S. Raymod, "The Cathedral and the Bazzar," Tim O'Reilly (Ed.). O'Reilly & Associates, Inc., Sebastopol, 1999.
2. R. Love, Linux Kernel Development, 3rd ed. ed., I. Person Education, Ed., Boston: Addison-Wesley, 2010.
3. B. Feiler, Writer, Agile programming - for your family. [Performance]. http://www.ted.com/talks/bruce_feiler_agile_programming_for_your_family, 2013.
4. V. Potdar and E. Chang, "Open Source and Closed Source Software Development Methodologies," in Feller, J. and Fitzgeralg, B. and Hissam, S. and Lakhani, K. (ed), ICSE 2004: Twenty Sixth International Conference on Software Engineering (with) Collaboration Conflict and Control: Proceedings of the Fourth Workshop on Open Source Software Engineering, Edinburgh, 2004.
5. J. Corbet, "How to Participate in the Linux Community," Linux Foundation, 13 May 2011. [Online]. Available: http://www.linuxfoundation.org/content/1-guide-kernel-development- process.
6. M. Bar and K. Fogel, Open Source Development with CVS, 3rd Edition ed., J. Duntemann, Ed., Scottsdale, Arizona: Paraglyph Press, 2003.
7. Black Duck Software, "The Ninth Annual Future of Open Source Survey," 15 April 2015. [Online]. Available: https://www.blackducksoftware.com/future-of-open-source.
8. D. Spinellis, "A Tale Of Four Kernels," CSE '08: Proceedings of the 30th International Conference on Software Engineering, pp. 381-390, 1 May 2008.
9. Coverity Scan, "Open Source Report 2014," Synopsis, Inc, San Fransisco, 2014.
10. Coverity, "2012 Coverity Scan Open Source Report," Coverity, San Francisco, 2013.
11. K. Beck, "Twelve Principles of Agile Software," 1 January 2001. [Online]. Available: http://agilemanifesto.org.
12. R. Goldman and R. P. Gabriel, "Innovation Happens Elsewhere: Open Source as Business Strategy," 25 April 2005. [Online]. Available: http://www.dreamsongs.com/IHE/IHE-28.html.
13. N. Brown, "Linux Weekly .Net," 8 June 2009. [Online]. Available: https://lwn.net/Articles/336224/.
14. S. McConnell, Code Complete, 2nd ed. ed., Walla Walla: Microsoft Press, 2004.
15. D. Wheeler, "Best Practices Criteria for Free/Libre and Open Source Software (FLOSS) (version 0.5.0)," Linux Foundation, Portland, 2016.
16. J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," Project MAC and the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge.
17. E. Gamma, Design Patterns: Elements of Reusable Object-Oriented Software, Zurich: Addison-Wesley, 1994.
18. D. Bovet, Understanding the Linux Kernel, 3rd ed. ed., Sebastopol: O'Reilly Media, Inc, 2006.

19. I. Bowman, "Linux as a Case Study: Its Extracted Software Architecture," ICSE, pp. 555-563, 1999.
20. P. Hinds, "Distributed Work," in Distributed Work, Cambridge, The Mit Press, 2002, pp. 382-481.
21. A. Silberschatz, Operating System Concepts, New Haven: Wiley, 2014.
22. M. Kerrisk, The Linux Programming Interface, San Francisco: no starch press.
23. A. Tannenbaum, "Modern Operating Systems," in Linux: A Case Study, Amsterdam, Pearson, 2015, pp. 713-857.
24. A. Ampatzoglou, "An Empirical Study on Design Pattern Usage on Open-Source Software," in ENASE 2010 - Proceedings of the Fifth International Conference on Evaluation of Novel Approaches to Software Engineering, Athens, 2010.