ISSN 2063-5346

A REVIEW ON ISSUES AND SOLUTIONS OF SOFTWARE PROJECT MANAGEMENT

B.Laxminarayan[1], Shailesh Kumar[2] , J.Somasekar[3]

[1] Ph.D. Scholar, Department of CSE, JJTU university.

[2] Department of CSE, Gopalan College of Engineering and Management, Bangalore.

[3]Department of CSE, Faculty of Engineering and Technology, Jain University, Global Campus, Bangalore.

[1]bdplaxman@gmail.com

Article History: Received: 12.05.2023 **Revised:** 25.05.2023 **Accepted:** 05.06.2023

Abstract

As the complexity of "Software Development' has increased since its inception from the middle of the 20th century, a new aspect has emerged in the modern days: "Improved Collaboration". Indeed, the increasing complexity of applications has demanded the usage of teams or groups to develop software - as individuals are unable to develop huge software systems with sufficient speed or quality. In software development, problem solving is critical. Many of the fundamental software development processes, from requirements analysis, specification, and design through testing and verification, may be seen as conventional problem-solving methods. This review will concentrate on collaborative or group problem solving research and development in the field of software development with the goal of identifying key outstanding challenges and future directions.

Keywords : Software Development, Requirements Volatility, Security Threats, Test Driven Development, Collaborative Problem Solving.

I. Introduction

Managing projects of various sizes and levels of complexity is a necessary component of doing business in the modern era regardless of the vertical industry. The software sector is characterized by new in-house software development and implementations, infrastructure-related initiatives, updates or upgrades, and the growing development of web-based solutions and mobile apps. The software industry is in a constantly evolving, driven in part by the globalization of a wide range of goods and services. The software industry undertakes a wide range of projects, each with its own set of obstacles, including the following:

- As a result of globalization, there is a lot of rivalry.
- Issues with outdated systems and infrastructure.
- Time to market pressures and adoption rates
- Software-as-a-service (SaaS) is gaining traction.

These characteristics may apply to many businesses but managing projects in the software industry can feel like being in a pressure cooker because of the speed at which technology evolves and competition

intensifies the pressure to deliver projects on time, on budget, and to the quality standards expected.

II. Objectives of the study:

The main objectives of the present review are as follows:

- To find the issues related to the software development programmers.
- To make understand the cause of related issues.
- To provide appropriate solutions to the software programmers.

Scope of the study:

This paper describes the main scope is an executable system dynamic review that was created to assist project managers in understanding the complicated effects of requirements volatility on software development projects. During the development process, changes to a set of criteria can happen at any time (Kotonya and Sommerville, 1998). These modifications can be made "while the requirements are being elicited, analyzed, and validated, as well as after the system has been put into service." Previously, the attitude was that requirements should be firm by the end of the requirements phase and that they should not alter after that. This viewpoint is now widely recognized as unrealistic (Reifer, 2000).

III. A Review on Causes of Issues and Issues faced by Software Development Programmers

The present review is based on various issues faced by software development programmers.

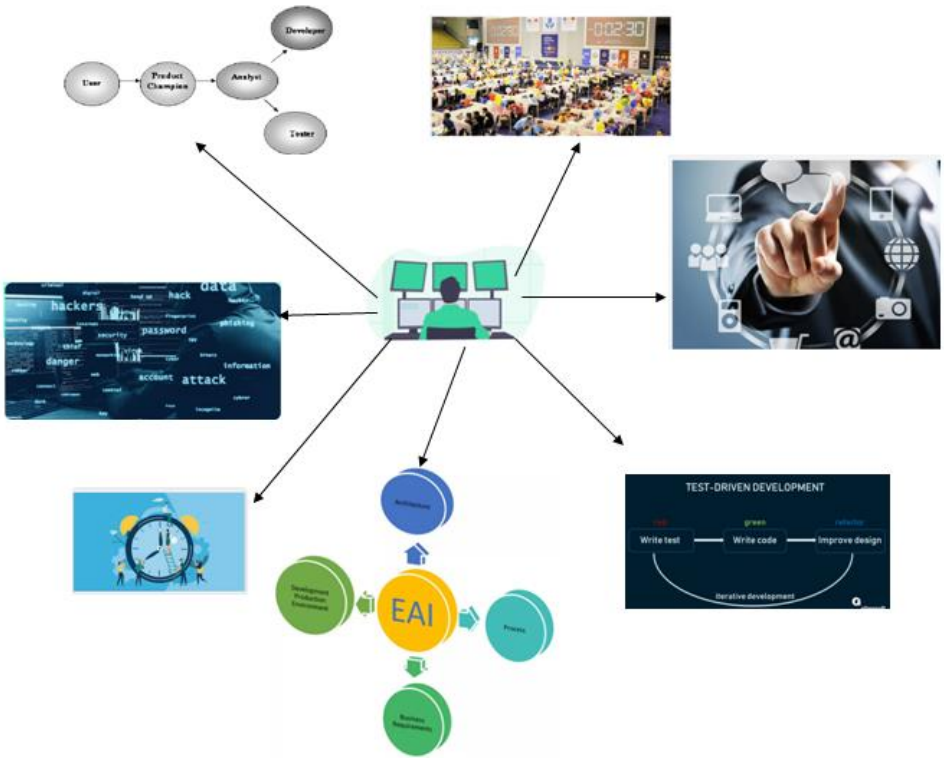

Figure1: Issues faced by software development programmers.

Requirements volatility:

During the software development life cycle, requirements can change. From the initial design phase through the execution phase, adjustments may be made. These changes that emerge during the development process pose a danger to the product's cost and quality, but they also present a chance to add value (Madachy and Tarbet, 2000;; Lin and Levary, 1989). The purpose of this article is to discuss requirements, their volatility, the sources of requirement volatility, and the influence of requirement volatility on project schedule, cost, performance, software quality, and maintenance. The goal of this paper is to go over some of the aspects of requirement volatility (Lane, 1998; Loconsole and Börstler, 2005, 2007; Lin et al., 1997).

High Competition:

Competition factors, whether local or worldwide, can have a significant impact on software firms' customer retention, pricing structures, client reach, service level agreements, and so on. As a result, it is critical to relieve pressure on team members in order to improve work performance. Project managers will need to engage closely with executives, business owners, and other stakeholders to flush out all aspects that may affect the success of software-related projects (Boltanski and Chiapello, 2005; Castells, 2000).

Open source software (OSS) has transformed large parts of the software industry in recent years, serving as an alternative to proprietary software and enabling radically new business models, organizational forms, and commons-based platforms as new value creation foundations (Fitzgerald, 2006; Krogh and Spaeth, 2007).

The fact that programmers must manage several logics driving the dominant rationale may therefore result in a set of conflicts (Bonaccorsi and Rossi, 2006).

Test driven development:

In fact everyone is aware that producing a product entails more than just writing code. Various iterations of testing are conducted throughout the project cycle to ensure that the actual product matches the expected outcome. It is also very typical to find difficulties/bugs during the testing phases, which necessitates retesting and repairing until the issues are fixed. Furthermore, project teams may find it more difficult to isolate difficulties, necessitating escalation to more senior IT staff/developers. Because project managers must apply excellent judgment to guarantee that all concerns are fixed prior to the systems going live. It goes without saying that the testing process is crucial for avoiding customer unhappiness and ensuring that no additional rework is required after going live.

TDD (test-driven development) is a software development method in which tests are written before any code is implemented. TDD is a test-first methodology that relies on the repetition of a short development cycle. Each new feature, according to it, begins with the creation of a test. Before writing enough production code to pass the test, the developer creates an automated test case. This test case will fail at first. The next stage will be to write code that focuses on functionality in order to pass the test. Following these steps, a developer refractors the code so that it passes all of the tests (Ivo et al., 2018).

Test-driven development has already been the subject of scrutiny. Turhan et al. [2], for example, conducted a thorough literature review on test-driven development that focused on both internal and external quality issues. Hundreds of articles have mentioned test-driven development in the recent decade, but just a few have reported empirically viable findings, according to the review.

Integration:

In software engineering, integration refers to the process of merging software components (also known as subsystems) into a single system. When compared to the sum of all independent systems, integrated systems perform better. What's more, a holistic application has a higher level of functional relevance. It's more straightforward to distribute and use. These days, integrating is more popular than ever. On the Internet, there are an increasing number of mergeable systems. As a result, maintaining contact with them is critical.

Many of these benefits were illusory in our experience with (Zelkowitz, 1990; Zelkowitz et al., 1988a, 1988b; Basili and Rombach, 1987; Duggan, 1988). While syntax-based editors make most programed building tasks go faster, there are a few popular constructs that cause issues and diminish the gains.

Threats to Security:

The fact that data is a valuable asset for practically everyone is undeniable. And some people, such as client, are willing to pay a high price for it. Clients undoubtedly rely on to protect their data from these attacks. And, believe it or not, that's a significant amount of pressure. Unfortunately, beginners frequently miss security flaws in their code and are unaware of the ramifications until after a security breach has occurred. A fresh programmer may find that overlooking security flaws, especially if programmer focus is on producing error-free code rather to ensuring that it is secure. However, hackers are constantly looking for ways to infiltrate code since they are aware of this flaw. No one can prevent someone from attempting to hack the code, but always be cautious and work harder to make it more difficult for them by securing it against popular hacking techniques. Here are some pointers:

a. Use parameterized queries for SQL injections.

An attacker may employ SQL injections to steal data such as a user's login credentials. To avoid this type of attack, use parameterized queries in the programming language using.

b. Ensure that your desk is safe.

An Attacker can be anyone. They do not often attack online. They may be someone from workplace, for example. For example, if fired an employee, he or she may decide to retaliate by modifying or stealing crucial project data using system. To avoid this type of attack, log off from any software are using once finished with it.

It is critical to understand the security risks associated with the software system you are developing in the software world (Ibrahim & Mamdouh, 2016). Software security is a philosophy that encompasses concepts, checklists, tools, processes, and methods, among other things (Gary, 2016). These are necessary for the architecture and design of software systems, as well as the coding and construction of software systems, as well as the verification and testing of software systems.

Basic security goals, such as confidentiality, integrity, and availability (the CIA trinity), as well as security risks regulating and monitoring activities and concepts like assets, threats, and vulnerabilities (Thamer & Mamdouh , 2016). Furthermore, even with some security measures in place, there are still hazards that a software may encounter that must be handled (Barry, 1991). In the software business, risk management solutions have played a critical role. It entails strategic software engineering methods that contribute to a more efficient software development process. As outlined by (Maruf et al., 2018), these approaches provide a disciplined atmosphere for successful decision-making by assessing actual and anticipated challenges in software development. Risk management is

the de facto method of dealing with risks (Thamer & Mamdouh, 2016). Risk management does not remove risks; rather, it is a method of identifying, assessing, minimizing, and dealing with them in a methodical manner. Its main goal is to identify potential difficulties before they develop so that they can be dealt with quickly and professionally. It should start as soon as possible and continue throughout the project's whole life cycle (Ibrahim & Mamdouh, 2016).

Ignoring Best Practices in Code Development:

It is typical for developers to skip code reviews or suppress defects in order to save time and fulfill deadlines. But don't forget that following a rigorous quality assurance approach is critical for a successful launch. So, if programmer, as a project manager, sees developers cutting corners in the development process, put a stop to it right away. Because it is important / necessary to apply excellent code development practices in order to satisfy the requirements sooner and more efficiently.

IV. A Review on Solutions to the Software Development Programmers

This paper is focused on collaborative problem solving because it's at the heart of collaborative software development: when someone develops software, they're designing a solution to a problem. Models for collaborative issue solving are still in the early stages of development. Various models have been devised by researchers, however they have not been thoroughly tested or implemented. Their use has been limited in both the industrial and educational sectors.

As a starting point, consider the models of Simon (1997) and Hohmann (1997). In the field of problem solving, Simon's (1960) influence has been seminal (Deek, 1997; Hohmann, 1997). Despite the fact that Simon's collaborative decision making model was limited to collaborative decision making, the similarities with collaborative issue solving (Huitt, 1992) make it a significant point of reference for the collaborative problem solving models we've discussed. Hohmann's (1997) model, on the other hand, is a good starting place because it is rather thorough and closely tied to some of Simon's most important work. Following is a list of qualities derived from Hohmann's (1997) and Simon's (1960) models.

Task identification: The goal of this task is to correctly identify the components of the problem, which can only be accomplished if the problem solution method is well comprehended. This stage also aids in the group's comprehension of the situation. According to Hohmann (1997), the members of the team, rather than an outside agent, are best suited to complete this duty.

Task distribution: The components of the problem solution should be allocated among the members of the group. This can be accomplished through voting, elimination, or direct assignment (Hohmann, 1997). Distribution by direct assignment may be simple if group members are aware of one other's abilities. If election is employed, it is critical that each group member be clear about which component they want to implement in order to secure a successful outcome. When there is only one component remaining and no one has been allocated to it, assignment by elimination occurs.

Coordinating Outcomes: Coordination is a continuous process in which each group member should take part. To enable members to interact effectively, groups require a tremendous lot of control and coordination (Finnegan & O'Mahony, 1996). This type of coordination increases the possibility of the group working together harmoniously to achieve the goal. Many problem-solving tasks require coordination, such as task distribution, subproblem integration, design debates,

and so on, hence collaboration is critical for successful development (Hohmann, 1997).

Integrating Solutions: Because the components identified during the solution design stage must be merged, an integration strategy must be created, starting with the solution integration order.

Plan Development: The behavior development plan is an integrated plan for all group members, not merely a collection of individual plans for each member (Simon, 1997).

Communication Strategy: Each member must be informed about the development plan (Simon, 1997).

Behavior Modification: Individual members must be willing to let the plan guide their actions (Simon, 1997).

V. Conclusion

Software development programmers' ultimate goal is to improve the software development process. To date, such applications have placed a greater emphasis on processes and technologies than on people. Current systems have similar limits, as well as corresponding chances for improvement. These shortcomings stem from "not comprehending the particular demands this class of software imposes on developers and users," according to a notable researcher in the field. The goal is to turn these flaws into research opportunities. Current collaborative problem solving models do not effectively or clearly address the qualities and requirements of group cognition, which is a significant drawback of current research. The restricted applicability of psychology and sociology in collaborative problem-solving models is another important constraint.

Collaboration in software development and problem solving will clearly speed up the software development process, allowing for faster, more cost-effective product delivery and more dependable creation of complex systems. Academic training in problem solving and software development can also benefit from appropriate collaborative environments. Because of the complexity and speed of application development, collaboration with coworkers, or group problem solving, is a required skill for today's software developers. The general result of present analysis of the literature is that merging perspectives and issues from collaborative problem solving, psychology, sociology, and collaborative software development can significantly advance the state of the art.

References:

1. Simon, H. A., The New Science of Management, Harper and Row, New York, 1960.
2. Simon, H. A., Administrative Behavior, Fourth Edition, The Free Press, New York, 1997.
3. Hohmann, L., Journey of the Software Professional, Prentice Hall PTR, New Jersey, 1997.
4. Finnegan, P., O'Mahony, L., "Group Problem Solving and Decision Making: an Investigation of the Process and the Supporting Technology", Journal of Information Technology, Vol. 11, Num. 3, September 1996, 211-221.
5. Huitt, W., "Problem solving and decision making: Consideration of individual differences using the Myers-Briggs Type Indicator", Journal of Psychological Type, Volume 24, pp. 33-44, 1992.
6. Deek, F., McHugh, J., Turoff, M., "Problem Solving and Cognitive Foundations for Program Development: An Integrated Model", submitted for review to the Journal of Cognitive Science, 2000.
7. Deek, F.P., Turoff, M., McHugh, J., "A Common Model for Problem Solving and Program Development", Journal of the IEEE Transactions on Education, Volume 42, Number 4., pp. 331-336, November 1999.

8. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D., 2001. Houston, D.X., 2000. A Software Project Simulation Model for Risk Management, Ph.D. Dissertation, Arizona State University.
9. Houston, D.X., Ferreira, S., Collofello, J.S., Montgomery, D.C., Mackulak, G.T., Shunk, D.L., 2001. Behavioral characterization: finding and using the influential factors in software process simulation models. Journal of Systems and Software 59 (3), 259–270.
10. Javed, T., Maqsood, M., Durrani, Q.S., 2004. A study to investigate the impact of requirements instability on software defects. ACM SIGSOFT Software Engineering Notes 29 (3), 7.
11. Jones, C., 1994. Assessment and Control of Software Risks. PTR Prentice-Hall, Inc., Englewood Cliffs, NJ.
12. Jones, C., 1998. Estimating Software Costs. McGraw-Hill, New York.
13. Kotonya, G., Sommerville, I., 1998. Requirements Engineering: Processes and Techniques. John Wiley and Sons, Ltd.
14. Lane, M.S., 1998. Enhancing software development productivity in Australian firms. In: Proceedings of the Ninth Australasian Conference on Information Systems (ACIS ’98), vol. 1, pp. 337–349.
15. Lin, C.Y., Abdel-Hamid, T., Sherif, J.S., 1997. Software-engineering process simulation model (SEPS). Journal of Systems and Software 38 (3), 263–277.
16. Lin, C.Y., Levary, R.R., 1989. Computer aided software development process design. IEEE Transactions on Software Engineering 15 (9), 1025–1037.
17. Loconsole, A., Börstler, J., 2005. An industrial case study on requirements volatility measures. In: Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC ’05), 8 p.
18. Loconsole, A., Börstler, J., 2007. Are size measures better than expert judgment? An industrial case study on requirements volatility. In: Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC ’07), pp. 238–245.