ISSN: 2633-4828 Vol. 5 No.4, December, 2023

International Journal of Applied Engineering & Technology

# THE STUDY OF COMPONENT BASED SOFTWARE INDUSTRY

**Praveen Kumar[1] and Dr. Kailash Patidar[2]**
[1]Research Scholar, Computer Sciences & Application, Dr. A.P.J. Abdul Kalam University, Indore
[2]Research Supervisor, Computer Sciences & Application, Dr. A.P.J. Abdul Kalam University, Indore
[1]Praveenkr1@gmail.com and [2]kailashpatidar123@gmail.com

## ABSTRACT
*Cost, timeframes, and quality are three of the most important factors in any software development process. Professional, technically competent labor and the constant evolution of technology account for the bulk of the price tag. Software projects can benefit greatly from schedule planning. Rapid software development is essential, as is meeting project deadlines. To compete in today's software business industry, the software development process has had to evolve along with the advancement of technology. Too many long-held assumptions about the software development process have been confirmed by experience and observation. However, in the long run, the cost-benefit analysis of the development process shows that not all forms of recursive reuse are worthwhile. Many large, well-known corporations have tried and failed to apply an improvement process strategy, while others keep trying but fail to fully consider all of the potential advantages. Many businesses still choose for laborious software engineering despite the fact that the upgrade's result is detrimental to their bottom line. The software business has seen a recent revolution due to a move toward a more modular approach, known as component-based software development. When different software components are joined, they form a whole with their own unique capabilities. The overall quality of a software product is equivalent to the quality of its constituent elements.*

*Keywords: Software, Development, Business, Quality, Product.*

## I. INTRODUCTION
The software industry includes businesses for development, maintenance and publication of software that are using different business models, mainly either "license/maintenance based" (on-premises) or "Cloud based" (such as SaaS, PaaS, IaaS, MBaaS, MSaaS, DCaaS etc.). The industry also includes software services, such as training, documentation, consulting and data recovery. The software and computer services industry spends more than 11% of its net sales for Research & Development which is in comparison with other industries the second highest share after pharmaceuticals & biotechnology.

The software industry has grown significantly during the last three to four decades. Automobiles, embedded systems, online commerce, and service industries like banking, telecommunications, and hotels all need software solutions. Cost, timeframes, and quality are three of the most important factors in any software development process. Professional, technically competent labor and the constant evolution of technology account for the bulk of the price tag. Software projects can benefit greatly from schedule planning. Rapid software development is essential, as is meeting project deadlines. The past of software development is not encouraging, since many missed deadlines can be seen across the field (Yu, 2009). Today, software quality is a central credo alongside cost and schedule for creating superior software solutions.

Since the 1980s, several software businesses have complained about issues with the software development process, claiming that a "software crisis" has taken place. Robert Glass says, "I look at my failure tales and see exception reporting, spectacular failures in the middle of numerous triumphs, a cup that is [now] almost full." While the software industry has made great strides so far, there is still room for improvement in the software development process (Glass,1998).

In the software development process, one of the biggest obstacles is making sense of and applying the necessary domain logic to the existing body of information (Hidalga et al, 2008).

A fast data center or cutting-edge hardware innovations won't make up for ignoring the work of a person to better the software development process (Brian, 2012).

Although open-source requires more skilled man power, it is the new doorway for the software business to serve consumer requirement, along with SaaS.

To compete in today's software business industry, the software development process has had to evolve along with the advancement of technology. Too many long-held assumptions about the software development process have been confirmed by experience and observation. However, in the long run, the cost-benefit analysis of the development process shows that not all forms of recursive reuse are worthwhile. Many large, well-known corporations have tried and failed to apply an improvement process strategy, while others keep trying but fail to fully consider all of the potential advantages. Many businesses still choose for laborious software engineering despite the fact that the upgrade's result is detrimental to their bottom line.

## II. LITERATURE AND REVIEW

**Dr-Rafiq Ahmad Khan et al (2022)** Security is one of the most critical aspects of software quality. Software security refers to the process of creating and developing software that assures the integrity, confidentiality, and availability of its code, data, and services. Software development organizations treat security as an afterthought issue, and as a result, they continue to face security threats. Incorporating security at any level of the Software Development Life Cycle (SDLC) has become an urgent requirement. Several methodologies, strategies, and models have been proposed and developed to address software security, but only a few of them give reliable evidence for creating secure software applications. Software security issues, on the other hand, have not been adequately addressed, and integrating security procedures into the SDLC remains a challenge. The major purpose of this paper is to learn about software security risks and practices so that secure software development methods can be better designed. A systematic literature review (SLR) was performed to classify important studies to achieve this goal. Based on the inclusion, exclusion, and quality assessment criteria, a total of 121 studies were chosen. This study identified 154 security risks and 424 best practices that help software development organizations to manage the security in each phase of the SDLC. To pursue secure SDLC, this study prescribed different security activities, which should be followed in each phase of the SDLC. Successful integration of these activities minimizing effort, time, and budget while delivering secure software applications. The findings of this study assist software development organizations in improving the security level of their software products and also enhancing their security efficiency. This will raise the developer’s awareness of secure development practices as well.

Filipe Arantes Fernandes et al (2022) Context: In the Software Engineering Education (SEE) context, virtual worlds have been used in order to improve learning outcomes. However, there is a gap in the literature in order to characterize the use of the Metaverse for SEE. Objective: the objective of this work is to characterize the state-of-the-art of virtual worlds in SEE, provide a research agenda to fill the limitations found, and propose an architecture of the Metaverse for SEE. Method: we conducted a systematic literature review, and we established 8 research questions that guided the study, as well as performed data extraction. Results: we report on 17 primary studies that deal mostly with immersive experiences in SEE. The results show some limitations: few SE topics are covered; most applications simulate environments and do not explore new ways of viewing and interacting; there is no interoperability between virtual worlds; learning analysis techniques are not applied; and biometric data are not considered in the validations of the studies. Conclusion: although there are virtual worlds for SEE, the results indicate the need to develop mechanisms in order to support the integration between virtual worlds. Therefore, based on the findings of the review, we propose an architecture of the Metaverse for SEE (MetaSEE). We hope that this work can motivate promising research in order to foster immersive learning experiences SE in the Metaverse.

Anil Jadhav et al (2022) Software development effort and cost estimation (SDECE) is one of the most important tasks in the field of software engineering. A large number of research papers have been published on this topic in

the last have decades. Investigating research trends using a systematic literature review when such a large number of research papers are published is a very tedious and time-consuming task. therefore, in this research paper, we propose a generic automated text mining framework to investigate research trends by analyzing the title, author's keywords, and abstract of the research papers. the proposed framework is used to investigate research trends by analyzing the title, keywords, and abstract of select 1015 research papers published on SDECE in the last have decades. We have identified the most popular SDECE techniques in each decade to understand how SDECE has evolved in the past decades. It is found that artificial neural network, fuzzy logic, regression, analogy-based approach, and COCOMO methods are the most used techniques for SDECE followed for optimization, use case point, machine learning, and function point analysis. • e NASA and ISBSG are the most used dataset for SDECE. • e MMRE, MRE, and PRED are the most used accuracy measures for SDECE. Results of the proposed framework are validated by comparing it with the outcome of the previously published review work and we found that the results are consistent. We have also carried out a detailed bibliometric analysis and metareview of the review and survey papers published on SDECE. this research study is significant for the development of new models for cost and effort estimations.

Filipe Arantes Fernandes et al (2019) This paper proposes to present an approach to obtain evidence about the influence of immersion in improving learning outcomes in Software Engineering. An immersive platform has been developed and a preliminary study with 6 subjects has been conducted to identify usability issues in an immersive application prototype to support the comprehension of Object-Oriented Paradigm theoretical concepts. Although the main problems are related to the affordances of the virtual world and the interaction between user and application, there was evidence that immersive technologies have the potential to support Software Engineering Education.

Dr-Rafiq Ahmad Khan et al (2022) Software security has become increasingly important because the malicious attack and other hacker risks of a computer system have grown popularity in the last few years. As a result, several researchers have examined security solutions as early as the requirement engineering phase. With the growth of the software business and the internet, there is a need to understand the security risks against each phase of the software development life cycle (SDLC). This study aims to empirically investigate and prioritize the risks that could negatively impact the software security aspects of SDLC in the context of global software development (GSD). To achieve the study objectives, we conducted an industrial empirical study to determine the impact of software security threats against each phase of SDLC. Furthermore, the fuzzy analytical hierarchy process (FAHP) was used to prioritize the list of software security risks against the SDLC. The results and analysis of this study provide a ranked-based decision-making framework, which assists practitioners in considering the most critical security risks on priority. The results show "improper plan for secure requirement identification, inception, authentication, authorization, and privacy," "lack of threat models updating," "lack of output validation," "lack of certification in the final release and archive," and "spoofing" as the top-ranked security risks of SDLC in GSD. In addition, the application of FAHP is novel in this domain as it is helpful to address multicriteria decision-making problems.

## III. SOFTWARE PROCESS IMPROVEMENT

SPI encompasses the following steps in the software development process:

(a) Examine and revise key performance indicators of a software development lifecycle.

(b) Find the worth of the approach's effective parameters by calculating how hard it is to use it to create software.

(c) Following the above examination, the best strategy for progress may be put into action. This SPI strategy redraws the conventional method of software development into one that is more malleable, adaptable, and trustworthy.

In spite of the time and work involved, the technique for software enhancement ultimately pays off. The effective software parameters might be reduced as a consequence of process improvements. There will be less chance of

flaws or bugs making through to the end user, less money spent on software upkeep, less time spent on quality problems, and fewer hidden expenses due to software delivery delays.

The technique for software enhancement is time-consuming and labor-intensive, but it pays off in the long run. As a consequence, more efficient procedures result in fewer requirements for software parameters. Also, it lessens the likelihood that problems will be passed on to customers, cuts down on the price of software upkeep, lessens the amount of time spent fixing quality problems, and lessens the indirect costs associated with delivering software late.

The Rout T 2002 (Terry, 2002)

SPI framework elements are described in figure 1.

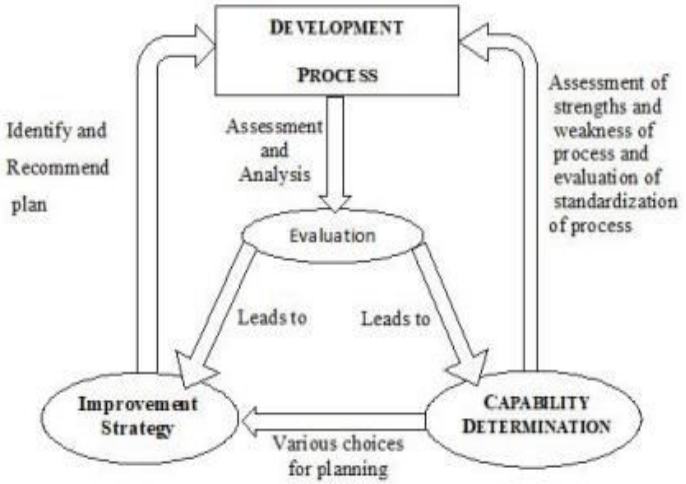


**Figure 1:** Elements of an SPI framework (Rout T, 2002)

Recently, SPI has been adopted as a regular practice by most major corporations. Contrary to popular belief, a significant amount of software development for major businesses is handled by small enterprises with a modest number of employees. It might be challenging for smaller companies to modify their SPI operations. Standard procedures for developing software for big corporations and small businesses are vastly different. It is typical for a smaller business to have fewer formal procedures and a looser structure. Even though it is proud and values originality, the SPI framework is sometimes criticized for being too bureaucratic and overthinking. As a result, process optimization is vital for businesses of all sizes. In small businesses, there may not be enough resources to fully execute an SPI structure. The execution of the SPI framework requires the commitment of enough resources, both monetary and human. As a result, it is important for software companies of all sizes to take into account the commercial rationale for SPI. Only once its advocates have shown that they can exert significant financial power are SPI projects authorized and put into action (Pino et al., 2008). Examining and implementing technical benefits (such fewer field defects, improved craftsmanship, cheaper maintenance costs, or shorter market time) demonstrates monetary leverage. Return on investment must be shown for SPI expenditures.

## IV. SOFTWARE ENGINEERING

The term "software engineering" refers to a group of practices in which an engineering mindset is applied to the creation and upkeep of software systems. By "technology," we mean everything from basic ideas to whole systems. To quote the IEEE standard definition of software engineering:

„*Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software* ,, (IEEE, 1990).

In this context, Philippe Kruchten recommends distinguishing qualities of software engineering over Civil, Electrical, and Mechanical engineering, as follows: -

- Software Engineering may be challenging to study in its theoretical form due to a lack of foundational ideas regarding the software development process.
- The effect of software modifications is hard to estimate, but they are encouraged.
- The fast pace of technological change prevents accurate evaluation of emerging technology.
- Note that when estimating the price of software, you should not include in time spent resolving bugs, making upgrades, or redesigning the system (Basili et al., 2008).

New software development methodologies, Open Source Software (OSS), Commercial Off-The-Shelf (COTS), and a more systematic synthesis and aggregation of international domains are some of the current issues in software engineering (Basili et al., 2008).

Structured, Object-Oriented Analysis (OOA), Agile, and Component-Based Software Engineering are just few of the various methods used to create software (CBSE).

Top-down design is the foundation of the structured programming paradigm. Here, the whole program is broken down into smaller components called modules. By separating code into modular chunks, structured programming makes it easier to fit more code into memory and allows for more code reuse. Research by Darcy and colleagues (2005) Mathematicians Corrado Bohm and Guiseppe Jacopini used only sequences, choices, and loops to prove the idea of structural programming.

The Object Oriented methodology is a powerful tool for addressing practical issues. The mechanisms for reusing code blocks, designs, and analyses are all provided by object-oriented programming. System design, class design, interface design, and database design are the four components that make up the OO approach's design process (Booch, 1996). Classes and objects play a central role in the OO methodology at all stages from analysis to implementation. Object-oriented languages like C++ and Java provide a variety of useful features, including encapsulation, inheritance, data binding, and polymorphism.

Agile software development is an iterative process that emphasizes incremental definition, design, and implementation and offers comprehensive testing and development (Sommerville,2010; Talby et al., 2006). One of the most well-known approaches of Agile software development is known as E Xtreme Programming (XP) (Auer and Miller, 2001).

## V. COMPONENT-BASED SOFTWARE ENGINEERING

Around the turn of the millennium, the term "component-based software engineering" (CBSE) began to circulate. Because software developers aren't happy with the results of object-oriented reuse analysis, this method was developed. Loosely connected software component integration entails the definition, selection, implementation, retrieval, and integration of such components into applications.

Through the use of integration mechanisms such as interfaces, pipes, and glue code, CBSE increases the degree to which software may be reused by building applications from pre-existing software components. Figure 2 shows the main steps of Sommerville's CBSE method (2010).

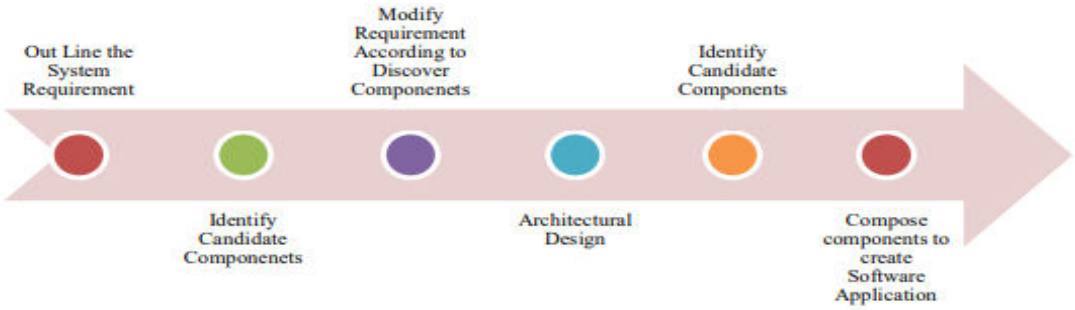


**Figure 2:** Component-based Software Engineering Process

## VI. BENEFITS OF COMPONENT-BASED SOFTWARE ENGINEERING

1. **Reduced Development Cost:** Component-based Through the use of software reuse principles, Software Engineering helps keep development costs down. Basically, it eliminates the necessity to create software from scratch. Consequently, this reduces waste and, by extension, costs.
2. **Reduced Development Time :** All software components are thoroughly tested and debugged before development even begins, which is why CBSE cuts down on development time. Therefore, while using software components, the time needed for testing and debugging the application is reduced.
3. **Easy To Manage Software Complexities :** All software components in component-based software engineering are pre-tested, certified, and documented, making it easy to handle the intricacies of software development. We can quickly detect and correct any discrepancies.
4. **Improved Quality of Software Development Process :** There are several opportunities for mistake detection and correction throughout the process of reusing software components. That's why the software we make is so reliable.
5. **Reduced Defect Density:** The defect density is obtained by dividing the total number of defects by the total number of lines of code. Because CBSE uses pre-tested and configured software components, the resulting applications have a lower defect density than those whose design did not make use of reusable software components.

## VII. CONCLUSION

Recently, a shift to a more modular approach known as component-based software development has sparked a revolution in the software industry. Software is made up of separate, specialized modules that are then combined. The overall quality of a piece of software is determined by the sum of its parts. The software sector relies on relatively new technology and is likely to be more aware of the need to innovate than other industries. It is not surprising, then, that the firms surveyed or studied as cases all seemed explicitly aware of the knowledge intensive services activities they needed and used. In particular, they were aware of the need to retain and build on in-house capabilities, and they recognised that purchased services were a source of new capabilities. Most firms had explicit mechanisms for absorbing and developing the new knowledge that the services brought them.

## REFERENCE

1. Khan, Dr-Rafiq & Khan, Siffat Ullah & Khan, Habib & Ilyas, Muhammad. (2022). Systematic Literature Review on Security Risks and its Practices in Secure Software Development. IEEE Access. 10. 5456-5481. 10.1109/ACCESS.2022.3140181.
2. Fernandes, Filipe & Werner, Claudia. (2022). A Systematic Literature Review of the Metaverse for Software Engineering Education: Overview, Challenges and Opportunities. 10.13140/RG.2.2.25341.64489.

3. Anil Jadhav, Mandeep Kaur, and Farzana Akter (2022) Evolution of Software Development Effort and Cost Estimation Techniques: Five Decades Study Using Automated Text Mining Approach. Hindawi Mathematical Problems in Engineering Volume 2022, Article ID 5782587, 17 pages https://doi.org/10.1155/2022/5782587

4. Fernandes, Filipe & Werner, Claudia. (2019). Towards Immersive Software Engineering Education. 7-8. 10.5753/svr_estendido.2019.8451.

5. Khan, Dr-Rafiq & Khan, Siffat Ullah & Azeem Akbar, Muhammad & Alzahrani, Musaad. (2022). Security risks of global software development life cycle: Industry practitioner's perspective. Journal of Software: Evolution and Process. 10.1002/smr.2521.

6. L. S. Adriaanse and C. Rensleigh, “Web of science, scopus and Google scholar,” 0e Electronic Library, vol. 31, no. 6, pp. 727–744, 2013.

7. p. Mongeon and A. Paul-Hus, “'e journal coverage of web of science and scopus: a comparative analysis,” Scientometrics, vol. 106, no. 1, pp. 213–228, 2016.

8. B. Nie and S. Sun, “Using text mining techniques to identify research trends: a case study of design research,” Applied Sciences, vol. 7, no. 4, p. 401, 2017.

9. B. Sigweni and M. Shepperd, “Feature weighting techniques for CBR in software effort estimation studies: a review and empirical evaluation,” in Proceedings of the 10th International Conference on Predictive Models in Software Engineering, pp. 32–41, Torino, Italy, 2014, September.

10. H. Rastogi, S. Dhankhar, and M. Kakkar, “A survey on software effort estimation techniques,” in Proceedings of the 2014 5th International Conference-Confluence the Next Generation Information Technology Summit, pp. 826–830, IEEE, Noida, India, September, 2014.

11. F. Gonzalez-Ladr ´ on-de-Guevara and M. Fern ´ andez-Diego, ´ “ISBSG variables most frequently used for software effort estimation: a mapping review,” in Proceedings of the 8th ACM/ IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–4, New York, USA, September, 2014.

12. H. Hamza, A. Kamel, and K. Shams, “Software effort estimation using artificial neural networks: a survey of the current practices,” in Proceedings of the 2013 10th International Conference on Information Technology: New Generations, pp. 731–733, IEEE, Las Vegas, NV, USA, April, 2013.

13. P. Faria and E. Miranda, “Expert Judgment in Software Estimation during the Bid Phase of a Project--An Exploratory Survey,” in Proceedings of the 2012 Joint Conference of the 22nd International Workshop on Software Measurement and the 2012 Seventh International Conference on Software Process and Product Measurement, pp. 126–131, IEEE, Nagoya, Japan, Octobar, 2012.

14. M. El Bajta, A. Idri, J. L. Fernandez-Alem ´ an, J. N. Ros, and ´ A. Toval, “Software cost estimation for global software development a systematic map and review study,” in Proceedings of the 2015 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), pp. 197–206, Barcelona, Spain, April, 2015.

15. M. Saroha and S. Sahu, “Tools & methods for software effort estimation using use case points model—a review,” in Proceedings of the International Conference on Computing, Communication & Automation, pp. 874–879, Barcelona, Spain, April, 2015.