

A Survey on Various Software Development Life Cycle Models along with Machine Learning Scope

Gagana H¹ and Dr P Devaki²

¹⁻²The National Institute of Engineering, Department of Information Science & Engineering, Mysuru, India
Email: gaganah2@gmail.com, devaki@nie.ac.in

Abstract—Information Technology is a power hub of resources with each passing day; as well, people's demand is increasing each day resulting in high demand for products and services, which leads to high expectation of delivery of services on time. The organizational strategy employs immense use of the internet and PC, providing a strong impact on competitors against the customers. Our approach here is primarily focused on reviewing a software Development Life Cycle (SDLC), starting from the development of software, management of the software, and these results in examining the development of the software through various models developed widely known as SDLC. Moreover, this review various existing SDLC models including the Agile model, Waterfall, V-shaped, Spiral Model, Waterfall Model and recent models like DevOps. This research review provides a critical analysis of these models along with a comparative analysis considering the various constraints. At last, this research review performs the review analysis of the integration of Machine Learning and SDLC model for the future research directions.

Index Terms— SDLC, DevOps, Software model, Machine Learning.

I. INTRODUCTION

Software Engineering constitutes an objective that produces maintainable software, on time and good quality software with an optimal budget. In general, software is defined as the collection of various actions to perform or execute a defined task. It can also be defined as the particular set of instructions which instructs the system to perform. Moreover, there are certain criteria to design software as listed below.

- User involvement
- Stakeholder satisfaction
- Management support
- Clear requirements
- Dealing with change
- Proper planning and monitoring
- Sound project management
- Proper technology and methodology

Furthermore, designing software is not so easy and can fail if not followed certain procedures; software failure can occur due to a listed number of reasons below:

- Badly defined requirements
- Frequent changing requirements

- Use of wrong technology
- Use of the wrong methodology
- Unmanaged risks
- Poor communication among stakeholders
- Inability to deal with project's complexity
- Poor project management
- Commercial pressures
- Unrealistic project goals.

Thus, it requires developing a process model that can enhance the lifecycle of software. This phenomenon is known as SDLC (Software Development Life Cycle). SDLC is a framework that inhibits the development of software with good potential in less amount of time, it is also used to describe software. SDLC is used to bridge the gap between system development and software development. In our paper, we have emphasized our study on the principles which are traversed into a system. However, the innovative strategies are deployed on leadership for building models and strategies from its initial phase to the development of the software, system development collaborated as the SDLC. Figure 1 shows the SDLC procedure that comprises certain blocks namely Requirement Analysis, Planning, Architectural Design, Software development and testing and deployment.

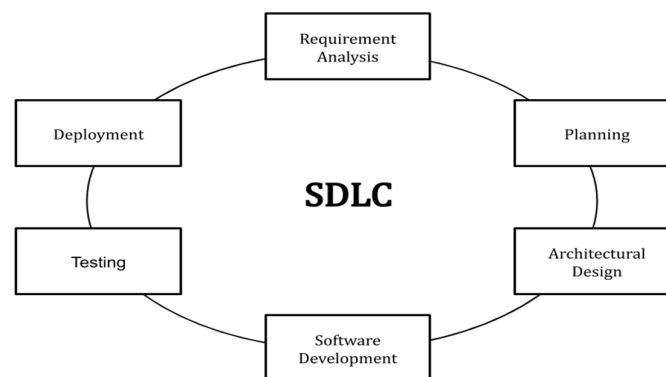


Figure 1 : SDLC

SDLC involves employing business ideas for building software applications, it involves various steps starting from planning, requirements gathering, deployment, and building, testing, and maintaining the software. Each model has a different approach to the SDLC lifecycle. An SDLC model exhibits the generic framework, which depicts the steps followed within the life-cycle framework. There exists a differentiation between a methodology and a model, a methodology describes a particular technique whereas a model describes the entire process to be exhibited. A model works as a descriptive approach whereas a methodology follows a prescriptive approach. We discuss various SDLC models ranging in their order of relevance or a specific type of project. Each model exhibits features relevant to a particular type of software project, consider the waterfall model is considered the best model for large-scale applications like enterprise applications and may not be a good model for website development.

SDLC models are divided into three types linear, iterative and a combination of linear and iterative both. A linear model works in a sequence, where upon completion of one phase provokes another phase working sequentially. An iterative model focuses on revisiting the stages another time in the future. The goal behind developing an iterative process is to put on a full stop at a particular stage.

SDLC is one of the important in the development of any kind of software as it optimizes the various services; over the years several SDLC has been proposed each possesses its advantages and disadvantages; however, every software developed adopts a particular SDLC. Motivated by the above phenomena, this research review presents the various SDLC model along with their advantages and disadvantages. Further, the research review provides a Machine Learning scope in SDLC.

This review work is designed as follows: The first section starts with the Background of software and SDLC; further, different SDLC phenomena and requirements are discussed along with the motivation and contribution

of research work. The second section extensively discusses the various SDLC model further third section includes the machine learning scope in SDLC

II. SDLC MODELS

Many SDLC models exist, the popular ones are stated below: waterfall model, spiral model, unified model, incrementing model, rapid application development, and the v and w models.

A) Waterfall Model

The waterfall model [1] combines the attributes of both a linear and an iterative model, hence the name known classical model. It is a primitive model widely used by most companies and government organizations. In this model the planning is involved in the early stages it illustrates all the flaws in the design before the development of the model. It incorporates good quality concerns, added provides good documentation of the model as well as the planning phase. This model initiates with requirements gathering, by incorporating system gathering to detailed design analysis, deployment of the model, coding, testing and maintenance.

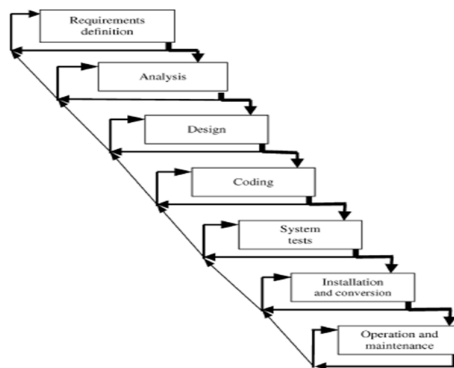


Figure 2 : Waterfall Model

Figure 2 shows the typical waterfall model that comprises various blocks namely requirement definition, Analysis, Design, Coding, System Tests, I&C (Installation and conversion) and O&M(Operation and Maintenance). Furthermore, the advantage and disadvantages of the model are described below:

Advantages:

- Easy deployment, and understanding of precipitation
- This model emphasizes defining the requirements before the design and designing the system model before coding.
- It recognizes the new invention and breakthrough to be achieved, well-maintained documentation, works better off on products developed and solves the problems associated with poor coordination among teams.
- Each phase in the model is processed individually and completed one at a time.

Disadvantages:

- This model follows an idealistic approach and doesn't go well with practicality.
- This model does not exhibit the iterative procedure followed by the exploratory development of a model.
- The requirements set up earlier in the project sometimes become unrealistic.
- As the software is developed in a later cycle of the project resulting in a delay in discovering serious issues and errors which may arise in the later phase of the project.
- Risk management cannot be incorporated as small changes in the documentation phase results in an increase in total expenditure.

Complex projects cannot be handled by this model, the cost incurred is high for small teams and projects.

B) Iterative Model

An iterative model starts with the same specifications and deploys the software, [3] this does not initiate the requirements phase. Here the development starts with specifying the requirements and then the development phase begins and then the implementation for just a part of the software, which is again revisited to determine

the further requirements. As the name suggests the process is iterated to create a new version of the software for each iteration of the model. Figure 3 Shows the Pictorial Representation of the given model.

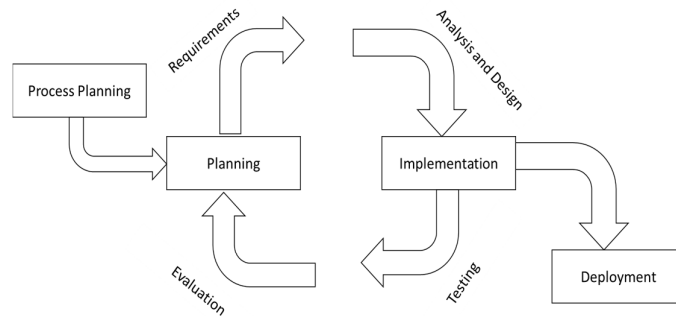


Figure 3: Iterative Model

Advantages:

- In this model, the software is built and improvised step by step, any failure or defect can be detected at the initial stage. This model doesn't allow defects to be carried out in the later stages. In this model, we also get relevant user feedback.
- In the iterative model, documentation time is minimized, whereas more concentration is given to the designing phase.

Disadvantages:

- In each iteration there exist no overlapping between the phases, designing the system architecture becomes more expensive and issues arise in the designing phase because the requirements gathered are not enough throughout the cycle.

C) V-shaped Model

The V-shaped model functions similar [2] to the waterfall model, by sequentially executing the processes. Each phase in the model needless to be completed before the other begins. In comparison with the waterfall, model testing is carried out rigorously in this model. Testing methods are developed in the initial stages of the lifecycle before the coding phase is initiated. Then comes the implementation phase, Like the waterfall paradigm, requirements are acquired in the first step. A testing strategy is designed before development begins, with the testing plan focusing on achieving the specific functionality identified during the requirements collecting process. This phase of high-end design focuses on system architecture and design. Integration testing is used in this architecture to ensure that all of the software components work together and collaborate

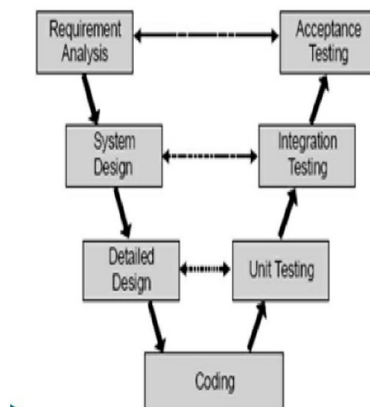


Figure 4 :V shaped Model

Advantages:

- This model is simple and convenient to use, each segment of the model has explicit takeaways.
- Due to rigorous testing in the initial phase, this model has gained wide success over the waterfall model during its entire lifecycle.
- Effective for small projects wherein the requirements gathered are understood with feasibility.

Disadvantages:

- Rigid nature just like the waterfall model, less flexible and difficult to adapt and is a little expensive.
- The software is built in the development phase, henceforth no early prototypes are developed initially to get a clear picture.
- This model definitively does not pave the way to solving all the problems.

D) Spiral Model:

The spiral model [4] is focused more on risk analysis and functions like that of the incremental model. This model is divided into four segments: Planning, Risk Analysis, Engineering and Evaluation. In this model, each phase is associated with each spiral and pass-through various iterations. In the planning phase initially, the requirements are gathered, and risk is assessed altogether. Each adjacent spiral builds the baseline spiral. In the planning phase, the requirements are gathered, in the next phase, the risk analysis process is taken into consideration to identify the risk and provide alternate solutions to it. A prototype is developed at the end of the risk analysis phase, Software or process is developed in the designing phase, as well as testing is done in this phase. In the evaluation phase, it allows the customers to evaluate the project output, which is transferred along with each spiral one after the other. in the spiral model, the angular component depicts the progress, and the radius depicts the cost incurred.

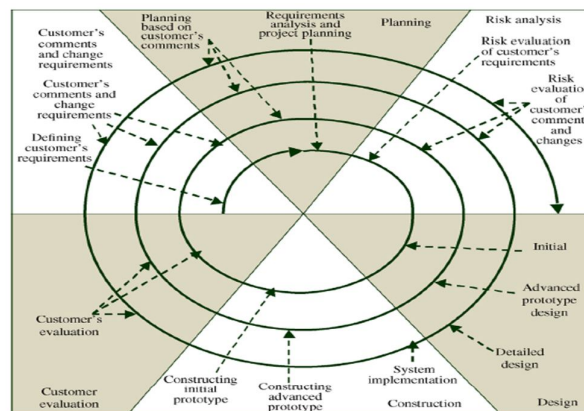


Figure 5: Spiral Model

Advantages:

- Risk analysis is done effectively, this model works effectively for large and complex projects. Software is developed in the initial phase of the SDLC.

Disadvantages:

- Model development cost is high, Risk analysis is carried out using great expertise. The completion of the project greatly depends on the risk analysis phase. This model works less efficiently for small projects.

E) Incremental Model:

The Incremental model is developed to handle the limitations of the waterfall model. In the initial phase, planning begins and in the end the development of the model along with mutual interactions in between. The main aim here is to deploy a system incrementally in small pieces at an incremental time. The software allows developers to make use of previous learning for the development in the initial phase or of the system [6].

Advantages:

- The business outcome is developed earlier in the developmental life cycle.
- The resources are utilized efficiently for proper incremental definition.
- Some changes can be incorporated between each request.

- Customer satisfaction is given a top priority, then to any linear approach. Problems are detected in the initial phases.

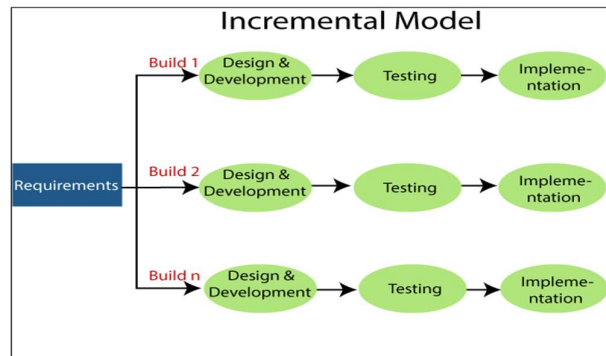


Figure 6: Incremental Model

Disadvantages:

- Documentation is necessary for this model; this follows a definitive procedure defined in increments focussed on a particular approach considering the dependencies on the feature.
- The functions and dependencies may create a problem if needed to be partitioned and require customer participation, unlike linear approaches.
- During the iterations, the problem of integration should be considered for the implementation.

F) Rapid Application Development (RAD)

This model is a functionally [7] equal to a subcategory of the functionally developed model. Since the prototype developed is evaluated by customer involvement, hence the resulting specification should be correct. In the Verification phase, specification, planning and design are encountered. In the development phase integration testing is necessary. A primary feature of rapid prototype development is the integration of the waterfall model with rapid prototyping in finding the customer's requirements

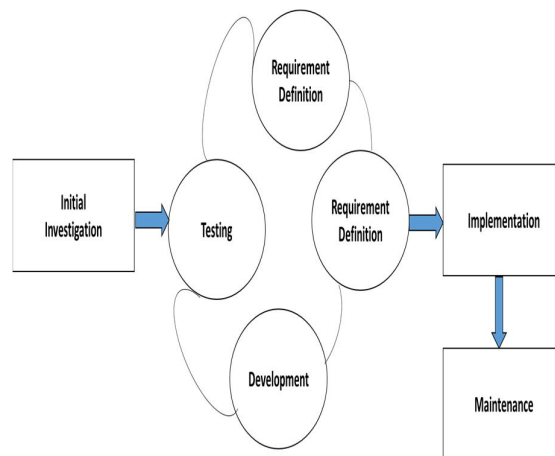


Figure 7: RAD Model

Advantages:

- RAD encourages the reusability of components, hence it reduces the development time to speed up the development process.
- The functions are modularized for each task to make it more convenient to work with.

Disadvantages:

- Skilled professionals are needed to handle large projects, which involve RAD.

- The development team should work with the customer collaboratively to complete the system model within a stipulated amount of time.
- Lack of commitment results in failing of RAD which is focused on an object-oriented approach and problematic in modularizing the project in which RAD doesn't work effectively.

G) Agile

This model is a traditional model, in which the scope change and features are incorporated by breaking up the project into smaller sub-divisions. The development is carried out in minimum-abbreviated time to accommodate the ongoing software changes to apprehend minor incremental changes. This model cannot be applied to large or complex projects which creates a problem because they give importance to communicating with the client in a real-time environment. Documentation is not given priority in this model which led to a substantial amount of documentation while de-emphasizing a formal process. This model is more suitable for Category 3 applications.

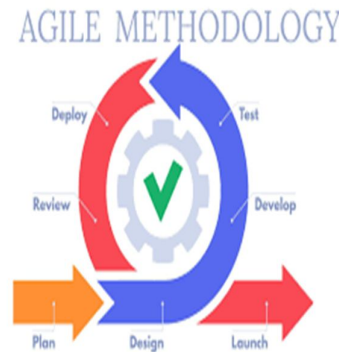


Figure 8: Agile model

H) Extreme Programming (XP)

This programming style is gradually evolving into a business champion that serves as a conduit for user-driven requirements and functionality. There is no initial design phase, which could result in greater costs if the requirements phase is not cooperative.

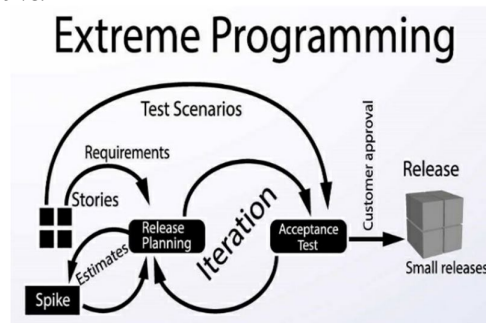


Figure 9: Extreme programming

To keep costs down, new requirements are acquired and accounted for in short, faster spirals, leading to the development of programmers working.

I) Joint Application Development (JAD)

This model accommodates integrated development with the client by his involvement in the entire process during the design and development phase encountered throughout the workshops (termed as JAD sessions). The probability of this model creates a scope in a system, which meets the client's requirements that is left unmanaged. Figure 10 shows the pictorial representation of Joint Application Development.

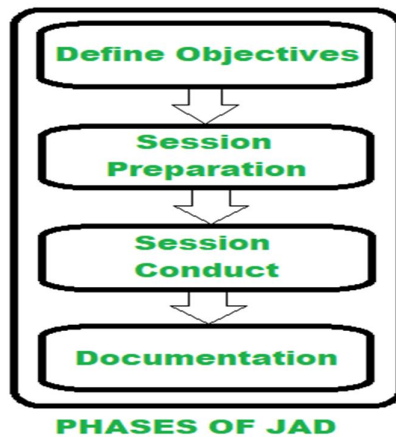


Figure 10: Phases of JDA

J) Lean Development (LD)

Lean Development (LD) is a model that works in pursuance with the logic of ‘80% today is better than 100% tomorrow’, Lean Development model delivers software or project with the limited specification. Scrum Development functions in a short-stipulated amount of time that works in short iterations, or sprints, and the growth is estimated daily. The scrum-based approach is suitable for small projects since it gives less importance to large or complex projects for a process-driven platform. As well the central server may lack team unity in large teams. RAD, as the united methodology framework is described here, in general perspective is much suited for category 1 and 2 functionalities, and is more comprehensive to Category 3 functionalities, which emphasize on least formal process-independent approaches.

TABLE I: COMPARATIVE ANALYSIS OF VARIOUS SDLC MODELS

Software Types	Complexity	Cost	Flexibility	Required Expertise	Release of product
V-shape Model	Medium	High	Highly flexible	Medium	IN phases
Waterfall Model	Simple	Low	Less Flexible	High	At the End
Prototyping Model	Complex	High	Less Flexibility	Medium	In Phases
Incremental Model	Simple	Low	High flexibility	High	In the phases
Agile Model	Medium	Low	Highly Flexible	High	In the phases

III. DEVOPS

A) DevOps Lifecycle

DevOps is a term that describes the dynamic interaction between development and operations [5]. From the beginning to the end of the product development process, the team and operational engineers practice it together. Previously, either the waterfall or Agile development models were used to create Dev-ops software solutions. Dev-ops was born out of the need to shorten the SDLC process to meet deadlines and produce high-quality, bug-free software. Dev-ops increased overall software production by combining features including a solid operating environment, super-fast delivery, constant collaboration, time optimization, and continuous innovation.

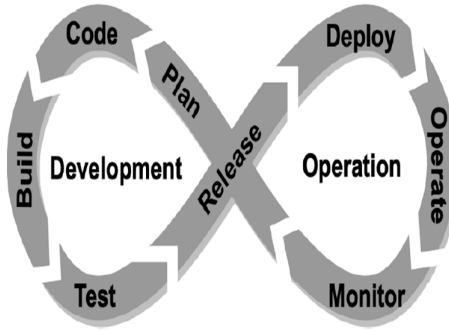


Figure 11: DevOps

DevOps is a software industry [11] practice of combining the main activities of producing and operating software applications through unified platforms. It is a mix of Development and Operations. This idea contains several significant phases since it describes the complete lifecycle of software programmers. We will go over several key DevOps Lifecycle Phases to help you grasp this crucial software integration idea. The DevOps movement highlights the important requirement for enterprises to be involved in continuous development and respond to changes more swiftly. The software development approach is known as "DevOps" combines development and operations to promote agility. DevOps improves the collaboration while making the communication quick for continuous delivery, getting regular updates, and hence promoting reliability, etc. [12].

B) DevOps challenges

Challenges faced by DevOps are diverse because they involve engineers and technical aspects, operations team have different precipitation of a particular problem in comparison with the development team. The operations team is focused more on ensuring the reliability and stability of the model. The development team looks for change and incorporates new innovative tools. This task is challenging to bridge a gap between the different teams. Hence, further focus is carried out the DevOps adapts the practice of fitting in the right people for a specific position. The right leadership is responsible to incorporate any possibility for a change. The true reason why choosing the appropriate DevOps team is necessary is because apart from technical skills and behaviour of engineers is mainly focused on automation of the tools and systems. A primary target to reach the DevOps fraction involves continuous deployment, delivery, and integration. This phase involves an automatic rollback of techniques which are less effective and that have functional issues. Any organization that possesses manual checks to test the quality of code through an automatic deployment of techniques. These examples inhibit DevOps scenarios that cause uncertainty issues needed to be addressed. All these challenges are tough to overcome in a large case scenario. The primitive silo structures last for many years and employees practice many non-agile scenarios.

IV. MACHINE LEARNING IN SDLC

ML has advanced significantly in recent years and is now used on a global scale. When building software systems for applications like speech/image recognition or automotive [14], ML has been considered crucial as a subset of artificial intelligence. Different SE challenges and activities have also been addressed using ML approaches. Most typically, machine learning has been used to anticipate defects, estimate work, and find patterns and similarities in source code. The ML techniques are primarily intended to tackle problems that are

difficult to understand by humans, resulting in misleading [15]. Several types of challenges can be classified as ML types. There are three types of machine learning: supervised, unsupervised, and reinforcement learning. When using ML for code smell detection, [16] investigated data balancing strategies and handled unbalanced dataset difficulties. [17] suggested a method for predicting unpleasant odors using design models like class diagrams. Code inspection/analysis is the category in which machine learning techniques are used for code reviews. [18], for example, developed an ML technique to automate code reviews for submitted code. The code/program similarity category is used to identify specific pieces of code that are similar across two or more software projects. They have developed an ML technique for reducing the amount of program similarity comparisons used to discriminate between original and pirated/cracked software. Writing requirements specifications is considered a human-centric work throughout the requirements stage. [19] and Jahan et al. [20] used machine learning techniques to deduce the most likely specifications and identify unexpected behaviours from a variety of artefacts. ML was used by [21] to identify unclear needs across domains. Predicting design patterns throughout the architecture and design stage is an important reverse engineering effort for improving software integrity. It is, however, prone to false positives and negatives [22]. The process of detecting patterns can be efficiently taught using ML [22], as the number of patterns is rapidly expanding due to their variances. [23] used machine learning to create test data automatically to enhance return on investment. [24] found that machine learning techniques perform much better than team estimates in estimating software maintenance work (human-centric).

V. CONCLUSION

Software plays a crucial role in the daily life of human beings; hence, the optimal model to develop the same becomes vital. Initially, traditional approaches were used in the mainstream until the development of agile methodologies. In SDLC, it is always believed that the traditional approach would be easily replaced by the new development. However, SDLC requires a dynamic approach that can integrate two distinctive methodologies. This research reviews the various SDLC model such as the Waterfall model, Agile Model, V-shape Model, Incremental Model, and Spiral Model; a critical review of each model is presented along with its pictorial representation, advantages, and disadvantages.

In recent years, we have seen a huge growth in Artificial intelligence and its domain like Machine learning and Deep learning. In the future, we plan to do a more in-depth investigation of the interaction between the ML and SDLC stages. To that end, we plan to include ML concepts like classification, regression, SVM, ANN, inductive logic, Bayesian network, or deep belief network in our search query. We feel that by doing so, we will be able to gain a more focused view of the state-of-the-art.

REFERENCES

- [1] Royce, Winston, "Managing the Development of Large Software Systems", Proceedings of IEEE WESCON 26, 1970.
- [2] Kevin Forsberg and Harold Mooz, "The Relationship of System Engineering to the Project Cycle," in Proceedings of the First Annual Symposium of National Council on System Engineering, October 1991: 57–65.
- [3] Craig Larman and Victor R. Basili, "Iterative and Incremental Development: A Brief History". IEEE Computer (IEEE Computer Society) 36 (6): 47–56. Doi:10.1109/MC.2003.1204375. ISSN 0018-9162. Retrieved 2009-01-10., June 2003.
- [4] Boehm B, "A Spiral Model of Software Development and Enhancement", ACM SIGSOFT Software Engineering Notes", "ACM", 11(4):14-24, August 1986.
- [5] Mohamed Sami Abd El-Satar" Software Development Life Cycle Models and Methodologies", 2012, <http://melsatar.wordpress.com>
- [6] Craig Larman and Victor Basili, Iterative and Incremental Development: A Brief History, IEEE Computer, June 2003
- [7] C. Melissa McClendon, Larry Ragout, Gerri Akers: The Analysis and Prototyping of Effective Graphical User Interfaces. October 1996.
- [8] A review paper on Dev-ops lifecycle:" Beginning and more to know".
- [9] Barry Boehm, "Spiral Development: Experience, Principles, and Refinements", edited by Wilfred J. Hansen, 2000.
- [10] Roger Pressman, Software Engineering: A Practitioner's Approach, Sixth Edition, McGraw-Hill Publication.
- [11] K. K. Aggarwal, Yogesh Singh Software Engineering 3rd Edition
- [12] Sanjana Taya "Comparative Analysis of Software Development Life" ISSN: 2229-4333 (print), ISSN: 0976-8491(online), vol.2ISSUE 4, Oct-Dec201.
- [13] S. Shafiq, A. Mashkoor, C. Mayr-Dorn and A. Egyed, "A Literature Review of Using Machine Learning in Software Development Life Cycle Stages," in IEEE Access, vol. 9, pp. 140896-140920, 2021, doi: 10.1109/ACCESS.2021.3119746.

- [14] F. Falcini, G. Lami, and A. M. Costanza, "Deep learning in automotive software," *IEEE Softw.*, vol. 34, no. 3, pp. 56–63, Feb. 2017.
- [15] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, no. 1, pp. 249–268, 2007.
- [16] F. Pecorelli, D. Di Nucci, C. De Roover, and A. De Lucia, "On the role of data balancing for machine learning-based code smell detection," in *Proc. 3rd ACM SIGSOFT Int. Workshop Mach. Learn. Techn. Softw. Qual. Eval. (MaLTesQuE)*, 2019, pp. 19–24.
- [17] N. Maneerat and P. Muenchaisri, "Bad-smell prediction from software design model using machine learning techniques," in *Proc. 8th Int. Joint Conf. Comput. Sci. Softw. Eng. (JCSSE)*, May 2011, pp. 331–336.
- [18] H. Lal and G. Pahwa, "Code review analysis of software system using machine learning techniques," in *Proc. 11th Int. Conf. Intell. Syst. Control (ISCO)*, Jan. 2017, pp. 8–13.
- [19] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar, "Inferring method specifications from natural language API descriptions," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 815–825.
- [20] M. Jahan, Z. S. H. Abad, and B. Far, "Detecting emergent behaviors and implied scenarios in scenario-based specifications: A machine learning approach," in *Proc. IEEE/ACM 11th Int. Workshop Modeling Softw. Eng. (MiSE)*, May 2019, pp. 8–14.
- [21] A. Ferrari and A. Esuli, "An NLP approach for cross-domain ambiguity detection in requirements engineering," *Automated Softw. Eng.*, vol. 26, no. 3, pp. 559–598, Sep. 2019, doi: 10.1007/s10515-019-00261-7.
- [22] A. K. Dwivedi, A. Tirkey, R. B. Ray, and S. K. Rath, "Software design pattern recognition using machine learning techniques," in *Proc. IEEE Region Conf. (TENCON)*, Nov. 2016, pp. 222–227.
- [23] D. Zhang, "Machine learning in value-based software test data generation," in *Proc. 18th IEEE Int. Conf. Tools With Artif. Intell. (ICTAI)*, Nov. 2006, pp. 732–736.
- [24] O. Malgonde and K. Chari, "An ensemblebased model for predicting agile software development effort," *Empirical Softw. Eng.*, vol. 24, no. 2, 2019.