Machine Learning in Software Development Life Cycle: A Comprehensive Review

Maryam Navaei and Nasseh Tabrizi
Department of Computer Science, East Carolina University, East 5th Street, Greenville, NC, U.S.A.

Keywords: Software Engineering, Software Development Life Cycle, Artificial Intelligence, Machine Learning, Machine Learning Algorithms.

Abstract: This research concludes an overall summary of the publications so far on the applied Machine Learning (ML) techniques in different phases of Software Development Life Cycle (SDLC) that includes Requirement Analysis, Design, Implementation, Testing, and Maintenance. We have performed a systematic review of the research studies published from 2015-2021 and revealed that Software Requirements Analysis phase has the least number of papers published; in contrast, Software Testing is the phase with the greatest number of papers published.

1 INTRODUCTION

As Software Engineering has become an essential industrial domain and software systems have become increasingly complex due to the performance requirements of their operating environment, the ability to design, develop, maintain, and adapt these systems has surpassed human comprehension alone (Elhabbash et al., 2019). Therefore, software systems need to autonomously adapt to changing environments to guarantee expected quality of service (Wan et al, 2019).

Artificial Intelligence (AI) has a long tradition in computer science. Since 1950 in its first three decades, the academy and industry worked diligently to reach to human level machine intelligence. In retrospect that was an overly optimistic expectation (Holzinger et al., 2018). Nevertheless, the need for more automation and intelligence have led to further advances in ML and AI. Researchers are increasingly applying ML and AI to remediate failures and inadequacies in the software systems and the SDLC (Khomh et al., 2018). ML is a branch of AI that uses data or former development experience to enhance the performance standards of software systems (Zhang et al., 2017, Zhu et al., 2018).

Traditionally, software systems are built deductively, by implementing the rules that administer the system performances as program code. However, with ML techniques, these rules are collected from training data. Conventionally, ML methods can learn a model's parameters automatically using training data and thus it can create models with respectable performance that can satisfy the software system requirements. ML has achieved great success in challenging real-world AI and data mining scenarios, such as object detection, natural image processing, autonomous car driving, urban scene understanding, automatic machine translation of human speech, and web search/information retrieval, among others (Gong et al., 2019, Chen et al. 2015, Peng et al., 2017).

ML enables computer systems to use data, examples, and experience without human involvement but to replicate some form of human intelligence. ML algorithms are utilized to help computer systems learn rules from data (England, 2018). These technologies will be optimized to be the core components in a variety of software-intensive systems. Recent advances in ML have fostered extensive interest in the Information Technology industry to integrate AI capabilities into software and services. Systems employing ML technologies, have distinctive characteristics moderately distinctive from those of the other software systems. Functionalities of ML based systems heavily depend on the quality of training datasets used to create the predictive models. Changing the training dataset has significant impact on learning results and thus on functional behavior of programs (Bird et al., 2017, Nakajima, 2018).

Navaei, M. and Tabrizi, N.
Machine Learning in Software Development Life Cycle: A Comprehensive Review.
DOI: 10.5220/0011040600003176
In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2022)*, pages 344-354
ISBN: 978-989-758-568-5; ISSN: 2184-4895

ML is used to continuously improve system performance and efficiency on a particular task by helping computer systems to learn from experience how to perform the job autonomously. ML plays a significant role in SDLC and has been used in variety of domains (Alloghani et al., 2020, Shafiq et al., 2021, Abubakar et al., 2020). For example, these domains include defect prediction, requirements elicitation etc. Therefore, ML is becoming one of the most important technologies used in SDLC. It is being applied to poorly understood problem domains where little domain knowledge currently exists for the humans to develop effective methodologies. These scenarios include data mining large databases containing valuable but undiscovered implicit regularities, and domains where software systems must adapt to changing conditions (Khomh et al., 2018). ML has become the preferred technique for developing practical Software Systems in Computer Vision, Speech Recognition, Natural Language Processing, Robot Control, Health Systems, Banking, Defence, E-Commerce and many more environments (Panichella and Ruiz, 2020, Bhatore et al., 2021). Conventionally, developing software systems necessitates on specifying the requirements in advance to define how the system should behave.

ML techniques are being used to accelerate SDLC and has been used in many areas including behavior extraction, testing, and bug fixing (Meinke and Bennaceur, 2018, DE-Arteaga et al., 2018). This has led to a new paradigm in technology invention. During the SDLC, the development team must spend significant time discussing how the system must operate to decide which features should be prioritized and which ones need to be eliminated. This Requirement Analysis needs to be followed by Design, Implementation, Testing and Integration, and Maintenance phases (Cetiner and Sahingoz, 2018).

By utilizing ML algorithms, software developers can accelerate the decision-making process based on the previous projects to create data-driven business decisions to ultimately help decrease the development risks and costs. This is invaluable for software companies. Since projects can go over costs and deadlines, coming up with a precise budget and schedule to develop software systems can be an overwhelming task. Therefore, accurate estimation of efforts is highly crucial for Software Project Planning. With the help of ML techniques, the development team can analyze data from past projects to provide a more precise budget estimate and delivery time frame (Yurdakurbann and Erdogan, 2018).

ML also has a great impact in decreasing the time spent on prototyping as well as the number of expert engineers required to develop the software. In Software Testing, with the help of ML algorithms, testers can come up with more accurate results to reduce the errors in the system. ML techniques also help accelerate the process of finding defects. Since clean code is crucial for effective software maintenance, large scale code refactoring is unavoidable.

Utilizing ML techniques, developers can review the code and maintain it automatically. Since maintenance is one of the most critical yet expensive phases of SDLC, this results in developing high quality Software Systems. Traditional SDLC can benefit from ML models for rapid prototyping, intelligent programming assistants, automatic analytics and error handling, automatic code refactoring, and precise estimates and strategic decision-making. Given the significant role that ML techniques and AI-based systems can play in the development of Software systems, it is very important for both the Software Engineering and ML communities to research and develop innovative approaches to address the advantages and disadvantages (Khomh et al., 2018, Abubakar et al., 2020). Although ML datasets for SDLC analysis are often poorly documented and maintained and do not have clear creation processes (Hutchinson et al., 2021) there are many advantages in utilizing ML techniques.

Because of the significant role of ML techniques in SDLC and their advantages, we have prepared an overall review to see the impact of data analytics and ML algorithms on each phase of the SDLC. We were interested to identify phases in which ML models are widely being utilized. Our goal was to see why researchers focused more on certain phases than others when it comes to utilizing ML methodologies. We also wanted to see if ML techniques can be applied to less popular phases as well.

To prepare our review, we collected and compiled a database of almost 150 journal articles and conference papers retrieved from ACM, IEEE and Springer digital libraries within the past five years. We created figures to demonstrate visually and summarize current research trends. Our research uncovers significant differences in the use of ML methodologies in various aspects of Software Engineering and the SLDC (e.g., Requirements Analysis, Design, Implementation, Testing and Maintenance). After a discussion of related work and our methodology, we discuss our findings for each

phase of SDLC. We then conclude with a summary of our work and suggestions for further research.

2 RELATED WORK

In this paper we review current research on applications of ML to different phases of SDLC and SE generally. Because of the popularity of incorporating ML techniques into SDLC, several prior researchers have conducted empirical studies of SE for data science. One study focused on differences between the development of ML systems and non-ML systems in various aspects of SE. That study used interviews to identify pain points from a general tooling standpoint to discover the challenges and obstacles of implementing visual analytic tools as none of the SE phases have a developed set of tools and methods (Kim et al., 2018, Giray, 2021). Another study focused on characterizing professional roles and practices regarding data science (Wan et al, 2019).

ML Software systems are challenging to test and verify. Sometimes the ML model may be incorrect even if the learning algorithm is executed properly due to inaccurate training data. Traditional testing techniques are insufficient for such systems. Therefore, it is crucial for Software Engineers to research and develop advanced ways to report these challenges (Khomh et al., 2018). Software Engineers focused on the challenge of verifying accuracy of systems built using ML and AI models and testing those systems.

Multiple previous literature reviews have been written on applying ML algorithms to each phase of SDLC (Talele and Phalnikar, 2021, Lima et al.,2020, Sobhy et al., 2021, Syaeful et al.,2017). We came across a publication that provides a broader context that discusses the relationship of ML techniques and its tools within SDLC stages which is related to our work to some extend however their review was performed for the period of 1991 to 2021(Shafiq et al., 2021). Nonetheless, our paper utilizes a systematic review methodology to study the impact of ML across the entire SDLC in the most recent publications (2015-2021) as opposed to focusing on a single phase as seen in most publications. Our research also makes an additional contribution by identifying the four most popular ML algorithms in overall SDLC. We studied those four techniques in each phase independently to observe their overall popularity based on each phase of life cycle.

3 METHODOLOGY

Initially, we intended to focus on publications for the past decade (2010-2020). Due to overwhelming number of publications, we finally decided to focus our review on the journal articles and conference papers published in the past five years. We realized the application of ML in SDLC considerably increased in the past ten years where it had a slow start from 2010 to 2015 and then significantly boosted from 2015 onward based on the number of publications, because recent advances in ML greatly impacted SDLS as ML techniques can change and update software systems.

We used different keywords and queries to retrieve our results from ACM, IEEE and Springer databases. Some queries gave us our desired results in IEEE and Springer whereas we had to modify our keywords and queries to retrieve relevant publications from ACM. Additionally, we found out that ACM would return significant number of irrelevant publications when using verbatim queries utilized in the other two libraries. Therefore, we had to apply more constraints when searching in ACM database. For instance, we had to include 'Artificial Intelligence' as a keyword to ensure our query retrieved relevant results.

There were situations where our queries returned similar or enhanced work in more than one publication, we treated each of those as a separate publication since each focused on a unique topic. Also, there were situations where one ML technique was utilized in multiple phases, we counted those separately for each phase. The latter scenario led us to study the four trendiest ML techniques in each phase furthermore to have a more focused observation.

4 CURRENT APPLICATIONS OF MACHINE LEARNING TO SDLC

ML plays a significant role throughout the SDLC. In this section we discuss the impact of applying ML in each phase of the SDLC:

- Software Requirements Analysis
- Software Architecture Design
- Software Implementation
- Software Testing
- Software Maintenance

We provide an overview of each phase of the SDLC and discuss current research trends in ML for each phase.

4.1 Software Requirements Analysis and Machine Learning

Software Requirements Analysis (also called Requirements Engineering) clarifies the specifications needed for a software system to satisfy the business requirements. High quality software development strongly depends on clearly defined Software Requirements (SR) that explains the needs and expectation of the software in a very detailed form (Navarro-Almanza et al., 2017). These requirements must be documented, measurable, testable, traceable, and related to the business needs.

Requirements Engineering (RE) plays an important role in the overall process of Software Development and consists of two main phases: Requirements Identification and Prioritization (Talele and Phalnikar, 2021). Requirements Analysis is critical to the success or failure of a Software system. Software Engineers must also perform Requirements Classification during the Analysis phase. Requirements Classification is cumbersome, time consuming, and difficult when performed as a manual process. RE consists of four parts: Requirements Elicitation and Discovery, Requirements Specification and Analysis, Requirements Validation and Requirements Management (Iqbal et al., 2018). However, after utilizing ML techniques, Software Engineers were able to automate the process of grouping the requirements into Functional Requirement (FRs) and Non-Functional Requirements (NFR) (Quba et al., 2021). RE is also one of the key factors in Software Quality (Panichella and Ruiz, 2020).

In SDLC, requirement analysis is the process of correct requirement gathering, the efficient examination of collected requirements, and clear requirement documentation. As software systems increase in complexity and scale, Requirement Engineering becomes a challenging issue that leads to increased development costs. This has led to engineers showing additional interest in automatic requirement analysis techniques which can lead to more accurate and rapid analysis of SR and reduction in development costs. ML methodologies are being used for Requirement Prioritization where engineers need to decide which set of requirements to be considered first and in requirement specification. ML algorithms are likely to classify and prioritize the requirements efficiently and how they can be evaluated (Talele and Phalnikar, 2021).

Researchers have shown a lot of interest in applying Supervised ML algorithms including Support Vector Machine, Naïve Bayes, Decision Tree, K-Nearest Neighbor and Random Forest when their focus is on one of the following broad categories (Gramajo et al., 2020):

- Detection of linguistic problems in requirements documents and artifacts written in natural language
- Classification of document content
- Requirement traceability
- Effort estimation
- Requirements analysis
- Failures Prediction
- Quality of and detection of business rules.

Techniques used in Software Requirements Specification (SRS) approach are Natural Language Processing (NLP) and Information Retrieval (IR) (Navarro-Almanza et al., 2017). ML provides approaches that use big data to enable a ML algorithm to learn from producing outputs, which would be difficult to obtain otherwise (Horkoff, 2019). RE plays a key role in the success of a project. Based on previous research that was conducted on 350 companies to understand project failure rates, 16.2% projects were completed successfully, 52.7% faced challenges and were partially completed. About 31% of the projects were never completed due to poor RE (Haleem et al., 2021).

One main limitation with public ML datasets is that there are a limited number of such datasets available for Requirements Analysis. One of the most used RE databases is the PROMISE repository, which is unbalanced and has only 625 classified requirements written in natural language (Iqbal et al., 2018). Despite limitations of current ML algorithms in recognizing and prioritizing requirements, including scalability, dependency, and complexity (Talele and Phalnikar, 2021), what prompts researchers to show continued interest in utilizing ML algorithms in SR classification is the significant role that these techniques play in helping developers document their software with more precision and validation. This makes the software system easier to understand and use (Quba et al., 2021).

4.2 Software Architecture Design and Machine Learning

The Architecture Design phase (also called the preliminary or general design stage) deals with converting the defined requirements from the Analysis phase into an implementable form. Software design has a major impact on software quality. Examples of bad design include anti-patterns, high dependency design, and massive source code files. These make SE tasks more difficult by exacerbating the challenges and expense of fixing software defects (Zhang et al., 2017, McIntosh and Kamei 2017).

This phase can be divided into three categories: Interface Design, Architectural Design and Detailed Design. Interface Design specifies interaction between a system and its users. In the Architectural Design, the specification of major system components, their responsibilities, relationships, and their interactions are defined. During Detailed Design the specification of internal elements of all major system components along with their properties, relationships and data structures are determined. Evaluation of system architecture is a turning point in the decision-making process. It aims at justifying the extent to which decisions made during Architecture Design satisfy a system's quality requirements. Specifically, when there are functional uncertainties and changing requirements. Architecture Evaluation facilitates the process of identifying and minimizing the design risks that save integration, testing and evolution costs in software systems (Sobhy et al., 2021).

Software Design should be clear and understandable to meet all system requirements. Since subsequent phases of SDLC significantly rely on the Architecture Design phase, many companies and researchers show interest in understanding the relationship between Software Design and Maintenance. Researchers have been working on introducing various ML techniques for efficient prediction of Software Design's impact on maintainability of a system (Elmidaoui et al., 2020).

Several ML techniques are being used in this phase on criteria such as predicting the design for a mobile application's user interface (UI), predicting the architecture for safety critical systems, web service anti-pattern detection, etc. For instance, design patterns can add complexity. This leads to an increase in maintenance and evolution efforts. Using ML for design pattern detection decreases the system complexity and results in increased understandability of the software's architecture and design (Mhawish and Gupta, 2019, Dwivedi et al., 2016, Dwivedi et al., 2016).

Code Smells, Call Dependencies and Lines of Code can be indicators of poor architecture and design (McIntosh and Kamei, 2021). Code Smells are not the errors in implementation; rather they are weaknesses in the design of part of the software system that can either impede the development process or increase the chance of system failure and errors in the future. Detecting Code Smells could lessen the work of developers, resources, and cost of development (Kaur et al., 2017).

Since detecting Code Anomalies requires refactoring approach, Support Vector Machine is the most effective ML algorithm used for this matter (Gramajo et al., 2020). Design patterns are a known reverse engineering technique to solve numerous problems in the design phase; they can be mined from source code of similar category software. Data mining from design patterns, which is based on supervised learning and software metrics, is another most popular research area in Design phase. ML techniques are being used to enhance pattern mining by excluding as many false matches as possible. ML algorithms such as Layer Recurrent Neural Network, Random Forest, Support Vector Machine and Boost are mostly used in this process (Navarro-Almanza et al., 2017, Giray, 2021).

4.3 Software Implementation and Machine Learning

The Implementation phase involves construction of the actual software system. Coding the system takes place in this phase and when it is complete, the result will be evaluated against the elicited requirements from the Analysis phase. Software defects are prevalent in software development and might cause several problems for users and developers alike. As a result, research has examined distinct techniques to diminish the impacts of these defects in the source code. One of the most prominent algorithms focuses on defect prediction using ML methods. This helps developers in managing these defects before they are commenced in the production (Esteves et al., 2020).

Code refactoring is one of the popular research areas for this phase. Refactoring changes the system in such a way that does not alter the external behavior of the code but will improve its internal structure. Software containing code smells indicates a violation of design and coding practices in SDLC. This issue can turn into a larger problem as the effort to remove the errors will increase exponentially if code smells are left unremedied (Gupta et al., 2019).

ML algorithms can precisely model the refactoring recommendations. Process and ownership metrics both play a significant role in the creation of highly effective models. In addition, models trained with data from heterogeneous projects generalize better and achieve great performance (Aniche et al., 2020). God Classes, Long Methods, Functional Decomposition and Spaghetti Code are the most frequently identified code smells (Paiva et al., 2017).

Supervised Machine Learning algorithms are known to be the most effective approach in predicting code refactoring, as these will help developers to make faster and more-educated decisions considering what to refactor. These ML techniques evaluated included Support Vector Machines, Naïve Bayes, Decision Trees, Random Forest, Logistic Regression and Neural Networks. Random Forest has emerged as the most accurate approach in predicting software refactoring using Apache, F-Droid and GitHub datasets (Sagar et al., 2021).

4.4 Software Testing and Machine Learning

In order to verify and validate Software Systems, engineers need to find a system's faults and defects. Software Testing is thus one of the most important phases in SDLC. The objective of Software Testing is to reveal existing faults, particularly during automated testing (Nasrabadi and Parsa, 2021). Due to constant changes to the codebase of a system, errors, failures and defects can happen in system behavior, there should be automated tests to identify these errors before a system is deployed. Improvements in system infrastructure by finding its faults could save up to third of its costs (Lima et al., 2020).

Testing helps developers ensure the behavior of a system is working as specified. Testing efforts must be predicted in advance to guarantee time efficiency, effort, and cost usage (Syaeful et al., 2017). In general, Software Testing is an inductive inference in which the tester determines the general properties of a software system by carefully studying the behavior of the system on a finite number of test cases (Zhu et al., 2018).

As software systems increase in complexity, it becomes more difficult to detect faults (Li et al., 2020). We can collect numerous kinds of data during software testing process, including execution traces, coverage information for test cases, failure data etc. ML techniques applied during Software Testing identifies the patterns in data that often feature about data generation process and will be used for decision-making. Experts need to provide inputs to transform raw test data into abstract test cases (Bhavsar et al., 2019).

Test cases are generated to provide test output data. Decision trees or induction rules are widely used in Software Testing especially in fault prediction. Models produced by these techniques are simpler to interpret compared to techniques that are more sophisticated such as neural networks or support vector machines. Support Vector Machine and Random Forest provide best prediction models for these datasets (the most widely used Testing dataset is from NASA) in fault prediction (Bhavsar et al., 2019, Zhu, 2020).

When testing software, there are considerable advantages to be gained from techniques which propose unusual interactions within a system. Techniques such as Random Testing, Fuzzing and Exploratory Testing have a disadvantage, however, in that the outputs of the tests need to be manually checked for correctness which adds additional effort for Software Engineers (Roper, 2019). Currently testing techniques and software reengineering are critical and most crucial in determining software usability (Banga et al., 2019, Herzig and Nagappan, 2015).

Due to the quantifiable nature of Software Testing, there are many datasets available. Likewise, the number of publications is also significantly higher compared to other phases.

4.5 Software Maintenance and Machine Learning

The process of maintaining a software system after it has been delivered, modifying, and updating it to correct defects is called Software Maintenance. This is the final phase of SDLC. Software Maintenance has become an important area in SE related to software quality. Consequently, predicting this characteristic in an accurate and timely manner is a crucial requirement for efficient management during the final phase of SDLC (Gupta and Chug, 2021).

Software Maintenance incorporates repair, preservation, and continuing optimization of a system. This phase has four different categories: Preventive Maintenance, Corrective Maintenance, Adaptive Maintenance and Perfective Maintenance (Sulaiman, 2005). As change is inevitable, engineers must develop techniques for evaluating, controlling, and making modification. Software Engineers need to be capable of predicting software maintainability in time, ensuring a constrained and optimum use of the available resources. A large number of ML

techniques, including hybrid techniques, ensemble techniques, and meta-heuristic techniques have been explored for Software Maintainability Prediction (SMP) (Alsolai, 2018, Gupta and Chug, 2021, Baskar and Chandrasekar, 2018). Yet researchers continue to seek improvement in this area.

A significant amount of time sometimes about 60-70% in SDLC total cost is dedicated to maintenance purposes (Gupta and Chug, 2021). As software systems are getting more complex and increasing in size, maintenance has become increasingly difficult. Therefore, software maintainability has turned into a serious challenge that companies are dealing with. Software maintainability is directly connected with the financial implementation and project success (Jha et al., 2019).

Software Maintenance is classified based on three viewpoints: intention-based classification, activity-based classification, and evidence- based classification. Software Maintenance involves bug fixing, upgrading, and enhancing the software program.

Many ML and AI techniques have been used in Software Maintenance. Yet, there are not sufficient datasets available as input data. Among available datasets, their size is not large enough to give accurate results. More work is needed in this area. Fuzzy logic is the mostly frequently used ML technique in Software Maintenance predictive models (Haleem et al.,2021). Other techniques include ML algorithms such as Soft Computing, Fuzzy Networks, Evolutionary Algorithm, Deep Learning and ID3 (Xin et al., 2018).

5 RESULTS

We include publications from 2015 to 2021 in this review.

The figure below summarizes how many journal articles and conference papers are published in ACM, IEEE and Springer for the period of 2015 to 2021 that focus on ML techniques applied in each phase of SDLC.

Table 1 lists the keywords we used to retrieve these publications and ultimately generate the figures to show the overall results. For some of the phases, we had to modify our search queries as it was a quite challenge to find more related and accurate results. Whereas some phases like Testing, had a very large number of publications.

Table 1: Searched Keywords for Paper Retrieval.

Keywords (Searched Queries)
Software engineering AND Machine learning
Software Development Life Cycle AND Machine Learning
Machine Learning AND Software System Requirements Analysis
Machine Learning AND Software Design
Machine Learning AND Software Implementation
Machine Learning AND Software Testing
Machine Learning AND Software Maintenance
Impact of Machine Learning in Software Maintenance
Impact of Machine Learning in Software Architecture
Impact of Machine Learning in Software Requirements
Software Requirements Analysis AND Random Forest Algorithm
Software Requirements Analysis AND Support Vector Machine
Software Requirements Analysis AND Naïve Bayes
Software Requirements Analysis AND Decision Trees
Software Design Phase AND Random Forest Algorithm
Software Design Phase AND Support Vector Machine
Software Design Phase Analysis AND Naïve Bayes
Software Implementation AND Random Forest Algorithm
Software Implementation AND Support Vector Machine
Software Implementation AND Naïve Bayes
Software Testing AND Random Forest Algorithm
Software Testing AND Support Vector Machine
Software Testing Analysis AND Naïve Bayes
Software Maintenance AND Random Forest Algorithm
Software Maintenance AND Support Vector Machine
Software Maintenance Analysis AND Naïve Bayes
Software Maintenance AND Decision Trees

As shown in Figure 1, Software Requirements Analysis has the least number of papers published in this topic; in contrast, Software Testing has the largest number of publications. The reason is partially because there are many datasets available for Software Testing due to its quantifiable nature as well as the ease of obtaining data for this phase.

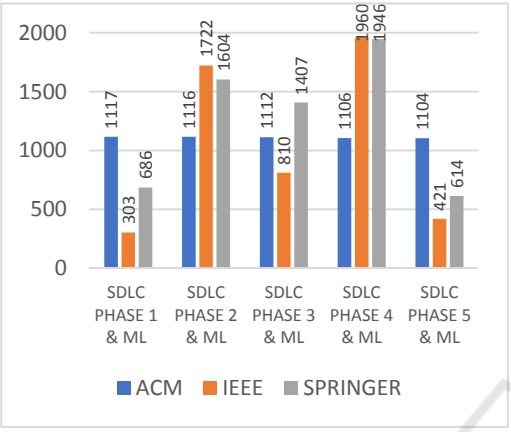

Figure 1: ML Applied in SDLC – Papers published from 2015 to 2021.

Hence it is much easier to utilize ML techniques in Software Testing than in Requirements Analysis.

In Figure 2, we narrowed our findings by selecting the four most used ML algorithms in all phases of SDLC (Random Forest, Support Vector Machine, Naïve Bayes and Decision Trees). Ultimately, we analyzed all the relevant publications done in ACM, IEEE and Springer in the same timeframe to see which ML methodologies are the most popular in particular phases.

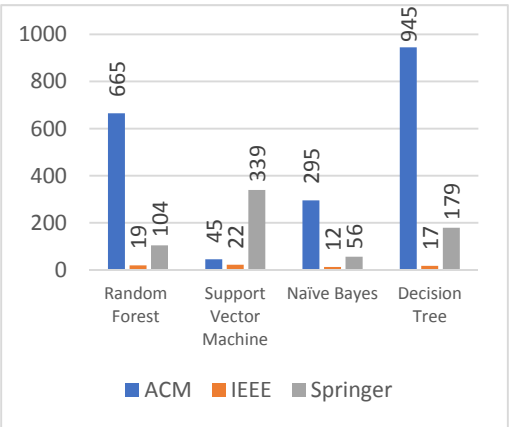

Figure 2: SDLC Software Requirements Analysis - ML Applied in SDLC – Papers published from 2015 to 2021.

Based on the results shown in Figure 2, Decision Trees are most popular algorithms among these four elected techniques. One reason is because Decision Trees are quick to create and help with eliminating dead ends. This decreases the probability of errors made in this phase that can affect subsequent phases as well. This technique can greatly simplify the SR gathering process and saves engineers significant time and budget resources.

In the process of SR, due to their exclusive characteristics, the appropriate data sets are extremely dimensional, sparse, and are mostly the outcome of ambiguous expressions and, consequently, show problematic challenges for data processing techniques (Borges et al., 2021). there are many repositories with duplicate code, which constitutes data inconsistency (Allamanis, 2018). Hence, more work needs to be done on creating more diverse and appropriate datasets for this phase.

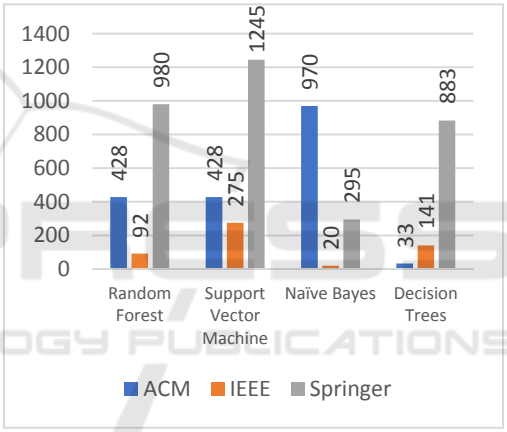

Figure 3: SDLC Software Architecture Design - ML Applied in SDLC – Papers published from 2015 to 2021.

Support Vector Machine (SVM) is one of the most popular ML techniques used in SDLC. It is a supervised learning algorithm used for both classification and regression problems. This technique is very accurate and robust in detecting design anomalies such as code smells also known as Bad Smells, Design Flaws, Anti Patterns and Code Anomaly (Zhang et al., 2006).

ML demands massive datasets to train on and these should be unbiased and have great quality which sometimes requires new data to be generated. We suggest applying ML algorithms more in Software architecture design as little work has been done and needs to be explored more (Borges et al., 2021).

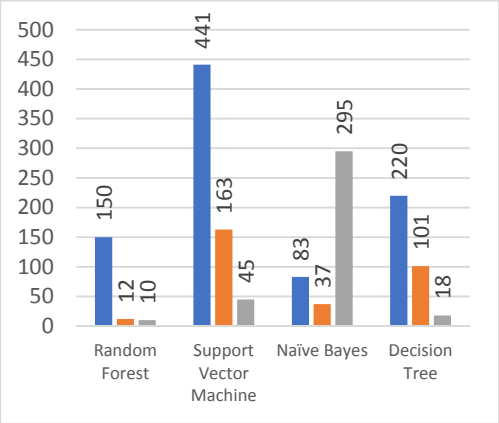

Figure 4: SDLC Phase Implementation - ML Applied in SDLC – Papers published from 2015 to 2021.

Again, Support Vector Machine is most frequent algorithm used in the third phase of Software Development Life Cycle known as Software Implementation. This technique is very popular in Design and Development of software applications.

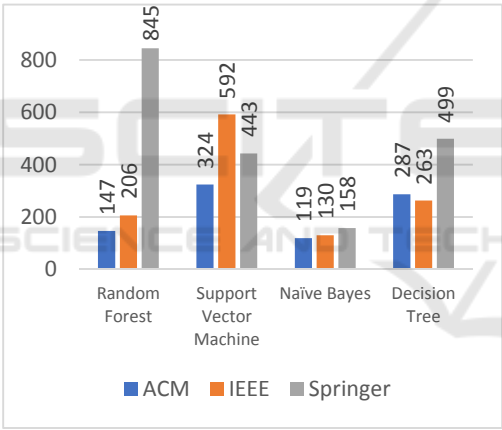

Figure 5: SDLC Phase Testing- ML Applied in SDLC – Papers published from 2015 to 2021.

As shown in Figure 5, both Random Forest and Support Vector Machine are popular techniques in Software Testing phase. However, SVM has slightly more publications. One reason for this is SVM is the better classifier to eliminate infeasible test cases saving time and costs in projects which is a huge achievement as testers spend more time and their resources on testing mobile and hybrid applications. ML helps testers to better understand user's needs and respond faster to the everchanging expectations by improving automation testing, reduced UI based testing, assisting in API testing, and improving accuracy.

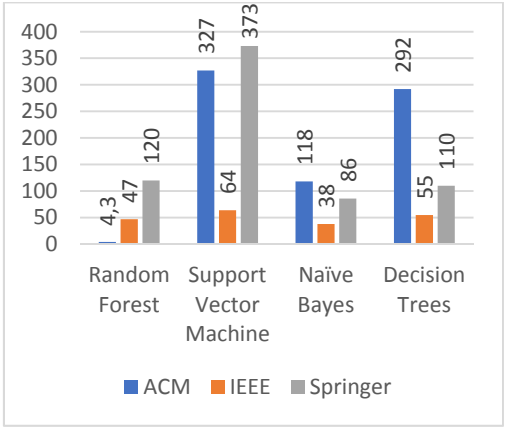

Figure 6: SDLC Maintenance - ML Applied in SDLC – Papers published from 2015 to 2021.

Data Scientists and Software Engineers have shown the most interest in applying Support Vector Machine in last phase of SDLC than other techniques. However, Decision Trees have high usage as well. Classification and refactoring approaches play a significant role when maintaining software systems. Available datasets are not sufficient to give conclusive results which provide information regarding the input data of the system (Jha et al., 2019) and more work need to be done to create new datasets to determine the accuracy of the precision and decreasing the complexity.

6 CONCLUSIONS

Machine Learning is becoming one of the most important technologies used in Software Development Life Cycle. ML approaches are being used for inadequately implied problem domains where little knowledge exists for humans to develop effective algorithms. ML has different types: Supervised Learning, Semi-Supervised Learning, Unsupervised Learning and Reinforcement Learning. Our study shows Support Vector Machine (SVM) is one of the most popular supervised ML algorithms in current SE research. It is being applied on all the phases of SDLC as it can be used for both classification and regression problems. While SVM is not considered the best performing algorithm in all cases, it remains among the most highly used ML methods. Reinforcement Learning is among the least mentioned ML algorithms in SDLC. Although it has been used in Software requirements analysis for understanding the relationships between requirements at different levels of abstractions a few times, it is an interesting topic that we can study on

more in terms of simplifying SE. Software Testing is utilizing ML techniques more than any other SDLC phase since there are a lot of datasets available to researchers.

Our future work aims at addressing the challenges outlined in Software Requirements Analysis, Architecture Patterns, Software Maintenance and ultimately creation of datasets for those phases. Also, future works could look at data mining, predictive design and modeling for different software applications including mobile applications.

REFERENCES

Abubakar, H., Obaidat, M. S., Gupta, A., Bhattacharya, P., Tanwar, S. (2020) - *Interplay of Machine Learning and Software Engineering for Quality Estimations – IEEE*

Allamanis, M. (2018) - *The adverse effects of code duplication in machine learning models of code – Research Gate.*

Alloghani, M., Al-Jumeily, D., Baker, T., Hussain, A., Mustafina, J., Ahmed J. Aljaaf (2020) - *An Intelligent Journey to Machine Learning Applications in Component-Based Software Engineering* - Springer. *Software Maintainability Metrics Prediction – IEEE.*

Aniche, M., Maziero, E., Durelli, R., Durelli, V. H. S. (2020) - *The Effectiveness of Supervised Machine Learning Algorithms in Predicting Software Refactoring – IEEE.*

Alsolai, H. (2018) - *Predicting Software Maintainability in Object-Oriented Systems Using Ensemble Techniques – IEEE.*

Banga, M., Bansal, A., Singh, A. (2019) - *Implementation of Machine Learning Techniques in Software Reliability: A framework – IEEE.*

Baskar, N. and Chandrasekar, C. (2018) - *An Evolving Neuro-PSO-based Software Maintainability Prediction – IEEE.*

Bhatore, S., Reddy, Y., R., Sanagavarapu, L. M, Chandra, S. S. (2021) - *Software Patterns to Identify Credit Risk Patterns – IEEE.*

Bhavsar, K., Gopalan, S., Shah, V. (2019) - *Machine Learning: A Software Process Reengineering in Software Development Organization – Research Gate.*

Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi B., Zimmermann, T. (2017) - *Software Engineering for Machine-Learning: A Case Study – IEEE.*

Borges, O. T., Couto, J. C., Ruiz, D., Prikllad nicki, R. (2021) – *Challeneges in using Machine Learning to Support Software Engineering – Research Gate.*

Cetiner, M., Sahingoz, O. K. (2020) - *A Comparative Analysis for Machine Learning based Software Defect Prediction Systems – IEEE.*

Chen, C., Seff, A., Kornhauser, A., and Xiao, J. (2015) - *DeepDriving: Learning affordance for direct perception in autonomous driving* - *IEEE.*

Dwivedi, A. K., Tirkey, A. , Rath, S. K. (2016) - *Applying software metrics for the mining of design pattern – IEEE.*

Dwivedi, A. K., Tirkey, A., Ray, R. B., Rath, S. K. (2016) - *Software design pattern recognition using machine learning techniques – IEEE.*

DE-Arteaga, M., Herlands, W., Neill, D. B., Dubrawski, A. (2018) - *Machine Learning for the Developing World – ACM.*

Elhabbash, A., Salama, M., Bahsoon, R., Tino, P. (2019) - *Self-awareness in Software Engineering: A Systematic Literature Review – ACM*

Elmidaoui, S., Cheikhi, L., Idri, A., Abran, A. (2020) - *Machine Learning Techniques for Software Maintainability Prediction: Accuracy Analysis – Springer.*

England, M. (2018) - *Machine Learning for Mathematical Software – Springer.*

Esteves, G., Figueiredo, E., Veloso, A., Viggiato, M., Nivio (2020) - *Understanding Machine Learning Software Defect Predictions – Springer.*

Giray, G (2021) - *A software engineering perspective on engineering machine learning systems: State of the art and challenges* - *Science Direct.*

Gramajo, M., Ballejos, L., Ale, M. (2020) - *Seizing Requirements Engineering Issues through Supervised Learning Techniques – IEEE.*

Gong, Z., Zhong, P., Hu, P. (2019) - *Diversity in Machine Learning – IEEE.*

Gupta, S., Chug, A. (2021) - *An Optimized Extreme Learning Machine Algorithm for Improving Software Maintainability Prediction – IEEE.*

Gupta, H., Kumar, L., Neti, L. B. M. (2019) - *An Empirical Framework for Code Smell Prediction using Extreme Learning Machine – IEEE.*

Haleem, M., Farooqui, M. F., Faisal, M. (2021) - *Cognitive impact validation of requirement uncertainty in software project development – Science Direct.*

Herzig, K. and Nagappan, N. (2015) - *empirically detecting false test alarms using association rules – IEEE.*

Holzinger, A., Kieseberg, P., Weippl, D., Tjoa, A. (2018) - *Current Advances, Trends and Challenges of Machine Learning and Knowledge Extraction: From Machine Learning to Explainable AI – Springer.*

Horkoff, J. (2019) - *Non-Functional Requirements for Machine Learning: Challenges and New Directions – IEEE.*

Hutchinson, B., Smart, A., Hanna, A., Denton, E. (2021) - *Towards Accountability for Machine Learning Datasets: Practices from Software Engineering and Infrastructure – ACM.*

Iqbal, T., Elahidoost, P., Lúcio, L. (2018) - *A Bird's Eye View on Requirements Engineering and Machine Learning – IEEE.*

Jha, S., Kumar, R., Son, L. H., Abdel-Basset, M., Priyadarshini, I., Sharma, R., Long, H. V. (2019) - *Deep Learning Approach for Software Maintainability Metrics Prediction*

Karim, M. S., Warnars, H. L. H. S., Gaol, F. L., Abdurachman, E., Soewito, B. (2017) - *Software*

metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset – IEEE.

Kaur, A., Jain, S., Goel, S. (2017) - *A Support Vector Machine Based Approach for Code Smell Detection – IEEE.*

Khomh, F., Adams, B., Cheng, J. , Fokaefs, M., Antoniol, G. (2018) - *Software Engineering for Machine-Learning Applications: The Road Ahead – IEEE.*

Kim, M., Zimmermann, T., DeLine, R. and Begel, A. (2018) - *Data scientists in software teams: State of the art and challenges – IEEE.*

Li, J. J., Ulrich, A., Bai, X., Bertolino, A. (2020) - *Advances in test automation for software with special focus on artificial intelligence and machine learning – Springer.*

Lima, R. , Da Cruz, A. M. R., Ribeiro, J. (2020) - *Artificial Intelligence Applied to Software Testing: A Literature Review – IEEE*

Meinke, K., Bennaceur, A. (2018) - *Machine Learning for Software Engineering: Models, Methods, and Applications – IEEE.*

McIntosh, S., Kamei, Y. (2017) - *Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction – IEEE.*

Mhawish, M. Y., Gupta, M. (2019) - *Software Metrics and tree-based machine learning algorithms for distinguishing and detecting similar structure design patterns – Springer.*

Nakajima, S. (2018) - *Quality Assurance of Machine Learning Software – IEEE.*

Nasrabadi, M. Z., Parsa, S. (2021) – *Learning to Predict Software Testability - IEEE.*

Navarro-Almanza, R., Juarez-Ramirez, R., Licea, G. (2017) - *Towards Supporting Software Engineering Using Deep Learning: A Case of Software Requirements Classification – IEEE.*

Quba, G., Qaisi, H. A., Althunibat, A., AlZu'bi, S. (2021) - *Software Requirements Classification using Machine Learning algorithm's – IEEE.*

Paiva, T., Damasceno, A., Figueiredo, E., Sant'Anna, C. (2017) - *On the evaluation of code smells and detection tools – Springer.*

Panichella, S., Ruiz, M. (2020) - *Requirements-Collector: Automating Requirements Specification from Elicitation Sessions and User Feedback – IEEE.*

Peng, H., Li, B., Ling, H., Hu, W., Xiong, W. and Maybank, S. J. (2017) - *Salient object detection via structured matrix decomposition - IEEE.*

Roper, M. (2019) - *Using Machine Learning to Classify Test Outcomes – IEEE.*

Sagar, P. S., AlOmar, E. A., Mkaouer, M. W., Ouni, A., Christian Newman (2021) - *Comparing Commit Messages and Source Code Metrics for the Prediction Refactoring Activities – Research Gate.*

Shafiq, S., Mashkoor, A., Mayr-Dorn, C., Egyed, A. (2021) - *A Literature Review of Using Machine Learning in Software Development Life Cycle Stages – IEEE.*

Sobhy, D., Bahsoon, R., Minku, L., Kazman, R. (2021) - *Evaluation of Software Architectures under Uncertainty: A Systematic Literature Review – ACM.*

Sulaiman, S. (2005) - *Viewing Software Artifacts for Different Software Maintenance Categories Using Graph Representations – Research Gate.*

Talele, P., Phalnikar, R. (2021) - *Classification and Prioritisation of Software Requirements using Machine Learning – A Systematic Review – IEEE.*

Wan, Z., Xia, X., Lo, D., Murphy, G. C. (2019) - *How does Machine Learning Change Software Development Practices? – IEEE.*

Xin, Y., Kong, L., Liu, Z., Chen, Y., Li, Y., Zhu, H., Gao, H., Hou, H., Wang, C. (2018) - *Machine Learning and Deep Learning Methods for Cybersecurity – IEEE.*

Yurdakurbann, V., Erdogan, N. (2018) - Comparison of machine learning methods for software project effort estimation – IEEE.

Zhu, H. (2018) - *Software Testing as a Problem of Machine Learning: Towards a Foundation on Computational Learning Theory – IEEE.*

Zhang, L., Tan, J., et al, D. H.. (2017) - *From machine learning to deep learning: progress in machine intelligence for rational drug discovery – IEEE.*

Zhang, X., Gu, C., Lin, J (2006) - *Support Vector Machines for Anomaly Detection – Research Gate.*

Zhang, X., Zhou, T., Zhu, C. (2017) - *An Empirical Study of the Impact of Bad Designs on Defect Proneness – IEEE.*

Zhu, Y, Chen, L, Zhou, H, Feng, W., Zhu, Q. (2018) - *Design and Implementation of WeChat Robot Based on Machine Learning – IEEE.*