

EXPLORING MACHINE LEARNING IN SECURING SOFTWARE REQUIREMENT SPECIFICATIONS: A SYSTEMATIC LITERATURE REVIEW

Vishok Kumar Singh¹ , Dr Ram Bhawan Singh²

¹Research Scholar, Computer Science and Engineering

Maya Devi University, Selaqui Dehradun

Email: vishok1982@gmail.com

²Professor, School of Computer Science and Engineering

Maya Devi University, Selaqui Dehradun

Rambhawan.singh@mdu.edu.in,

Correspondence mail: vishok1982@gmail.com

Abstract

Securing Software Requirement Specifications (SRS) is a critical step in ensuring robust and secure software systems, as vulnerabilities at this stage can propagate throughout the software development lifecycle. This systematic literature review explores the application of machine learning (ML) techniques in enhancing SRS security, focusing on identifying, mitigating, and predicting vulnerabilities and inconsistencies in requirement specifications. The study synthesizes recent advancements in the field, including the use of natural language processing (NLP) for extracting and analysing security requirements, predictive models for vulnerability detection, and hybrid approaches combining traditional security frameworks with ML. Key findings highlight the potential of ML to automate and improve the accuracy of security analysis in SRS while addressing challenges such as data scarcity, model interpretability, and domain-specific complexities. By identifying gaps in existing research, this review provides insights into emerging trends and proposes directions for future studies to advance the integration of ML in securing SRS. This work contributes to bridging the gap between academic advancements and industrial practices, paving the way for more secure software development processes.

Index terms Software Requirement Specifications, Machine learning (ML) techniques, natural language processing (NLP), SRS security

Introduction

Each and every software project must adhere to the security criteria that are outlined in the Software criteria Specification (SRS) of the Software Development Life Cycle (SDLC). Poorly written requirements that aren't in sync with the business logic of the software project, a disorganized and incomplete set of requirements, requirements that are outside of the project's scope, and requirements that are out of date are known to cause schedule delays and higher costs for development and maintenance [1]. It is possible that the cost of the project will grow by a factor of ten to two hundred times in the long run if security needs are not taken into consideration during requirement engineering and SRS [2].

Insufficient security consideration will result in low-quality software products that fail to meet software development goals [3]. The software development life cycle (SDLC) requires the collection and analysis of a document known as a Software Requirement Specification (SRS). This document acts as guidance for the next phase of the SDLC. After receiving the input via the statement of the issue, the SRS document is then created through the process. Each and every software development life cycle (SDLC) model is required to adhere to the six stages, which include requirement collection and analysis, design, coding, implementation, testing, deployment, and maintenance. The "requirement gathering and analysis" phase compiles the business requirements [3]. It is examined whether the collected requirements can be included in the system to verify their validity. Software Requirements Specifications (SRS) is the first step of the Software Development Life Cycle (SDLC), which measures the development of software. A collection of material that has been defined, standardized, and structured is known as SRS. Requirements that are associated with a software development project [18] [19]. It is important that the SRS document be created in a proper format.

The phase of demand collection and analysis has to include the feasibility study as a component or components. According to the criteria of excellent requirement characteristics, the requirements should serve the objective of the creation of the software product, which means that they should fulfil the requirements requirement. For the purpose of satisfying the customer's viewpoint, the collecting of requirements must contain both Functional Requirements (FR) and Non-Functional Requirements (NFR), which, when combined, will give a higher level of security in SRS. FRs are functions that are associated with the technical functioning of the system and define the interactions that take place between the system and the environment in which it operates. The importance of quality standards, such as performance, safety, security, dependability, and maintainability, is sometimes overlooked by FRs. In the process of designing the system, the implementation of the functional requirements takes place. NFRs are connected to the characteristics of quality. The implementation of NFRs occurs throughout the process of system architecture. The NFRs are constraints that are present in the process of system design or implementation. It is common practice to disregard non-functional requirements (NFRs) since they are quality requirements. The criteria may be infeasible, out of date, or not in sync with present circumstances. In many cases, non-functional requirements (NFRs) cannot be tested, and they are also incapable of being verified [2]. It is [5]. NFRs are supposed to specify the behavior that is anticipated from the software product or system.

Research Questions

- How has machine learning been applied to secure SRS in software engineering?
- What are the key challenges in securing SRS using ML techniques?
- Which ML algorithms or methods are most commonly used for securing SRS, and why?
- How effective are machine learning techniques in mitigating security risks in SRS?

- What are the emerging trends and future research directions in using ML for securing SRS?

Extracting and synthesizing the data

Table 1: Experts Contributions

Goal	ML Algorithm	Requirements Document	NFRs Addressed	ML Approach	Evaluation Methods & Results	Authors
Detect ambiguities in SRS	BERT (Deep Learning)	Public dataset (200 SRS documents)	Security, Reliability	NLP-based classification and tokenization	Precision: 87%, Recall: 89%, F1-Score: 88%	Amelia, O.
Classify security-related NFRs	Random Forest	Industry dataset (150 SRS)	Security, Usability	Feature extraction + supervised learning	Accuracy: 91%, Error Rate: 9%	Patil, V., &Jadhav, P.
Identify inconsistent security policies	K-means (Clustering)	Academic dataset (50 SRS)	Security	Clustering-based anomaly detection	Consistency improved by 20%	Hudaib, A., Salem, H., & Al-Tarawneh, K.
Evaluate redundancy in security NFRs	LSTM (Deep Learning)	Public dataset (100 SRS)	Security, Performance	Sequential modeling using NLP	Redundancy reduced by 18%	Alqurashi, S. S., Ray, I., &Malaiya, Y.
Predict vulnerable requirements	Decision Tree	Open-source projects (300 SRS)	Security	Rule-based classification + supervised ML	Detection Accuracy: 85%, False Positive: 12%	Ontiveros, J. A.
Generate secure NFR recommendations	GPT-based Model	Mixed dataset (200 SRS documents)	Security, Maintainability	NLP for contextual requirement	Recommendation Relevance: 90%	Mattioli, J., Awadid, A., &Sohier,

				generation		H.
Identify risks in system requirements	Support Vector Machine	Government projects (120 SRS)	Security, Privacy	Risk classification using SVM	Risk Identification Accuracy: 92%	Kolhe, K. P., & Jima, B. A.
Resolve ambiguities in SRS language	Naive Bayes	Public dataset (80 SRS documents)	Security	Statistical NLP	Precision: 80%, Recall: 78%, F1-Score: 79%	Mishrif, A., & Al Qamashoui, A.
Analyze compliance with security norms	XGBoost	Enterprise dataset (250 SRS)	Security, Compliance	Feature-based classification	Compliance Error Reduction: 25%	Guendouzi, B. S., & Ouchani, S.
Automate security NFR categorization	Transformer (Deep Learning)	Open-source projects (200 SRS)	Security, Usability, Scalability	NLP-based deep learning model	Accuracy: 93%, Scalability Improvement: 30%	Liu, J., Xiao, W., & Cheng, H.

Software Requirement Specification

The Software Requirement Specification, often known as SRS, is an exhaustive document that specifies the needs of a software system, both functional and non-functional. A basis for software development, it ensures that all stakeholders, including developers, customers, and end-users, have a common understanding of the system's goals and scope. It acts as a foundation for software development [4]. The program Requirements Specification (SRS) is like a contract between the client and the development team. It spells out everything about the program, such as how it works, how the system should behave, performance goals, and any limits. A well-thought-out software requirements specification (SRS) is an important part of the software development lifecycle (SDLC) because it reduces the chance of misunderstandings, stops scope creep, and clears up any confusion. The introduction, system overview, needs, and assumptions or dependencies are usually included. Non-functional requirements address characteristics such as security, scalability, stability, and usability, while functional requirements outline what the system is supposed to perform, such as processing inputs or creating outputs. Functional requirements are sometimes known as "functional requirements." The SRS is an essential step in detecting possible risks and vulnerabilities at an early stage of the development process [8]. This is from the point of view of security in general. If there are any unclear or incomplete requirements in the software requirements specification (SRS) or if security concerns are not taken into account, they

could all lead to major flaws in the final software product. For this reason, including security concerns in the SRS is very necessary in order to develop systems that are resistant to external attacks. Manual reviews, checklists, and templates for security requirements are some of the traditional ways that have been used in the past to handle security concerns in SRS. On the other hand, these procedures often require a significant amount of time and are prone to human mistake, particularly with the rising complexity of software systems. Through the course of the last several years, the use of cutting-edge technology like machine learning (ML) has emerged as a potentially useful approach for improving the security of SRS. Machine learning models are able to assess textual requirements by using methods like natural language processing (NLP). This allows them to find inconsistencies, ambiguities, and possible security risks. Predictive machine learning models that have been trained on historical data can also guess the risks that might come with certain needs, which lets us take preventative steps to lower those risks. Despite these breakthroughs, research remains ongoing in certain areas. These problems include the lack of data, the interpretability of models, and the alignment of machine learning predictions with industry standards [7]. The program Requirements Specification (SRS) is not only a technical document; rather, it is an artifact that is both dynamic and developing, reflecting the ever-changing requirements of users as well as the environment in which the program functions. There is no possible way to overestimate the significance of having a safe, transparent, and all-encompassing SRS in light of the increasing complexity of current systems. It is the foundation upon which successful software development projects are formed, and it ensures that security is included in the system from the very beginning. In table 1, focusing on the impact of Software Requirement Specification (SRS) on software development:

Table 2: SRS Behaviour

Aspect of Software Development	Role of SRS	Impact of a Well-Defined SRS	Impact of a Poorly Defined SRS	Source
Project Planning	Acts as the foundation for defining scope, timeline, and resource allocation.	<ul style="list-style-type: none"> - Clear goals and scope. - Accurate timeline and cost estimation. 	<ul style="list-style-type: none"> - Ambiguous goals. - Inaccurate planning, leading to delays and cost overruns. 	[19]
Requirement Clarity	Ensures functional and non-functional requirements are explicitly stated.	<ul style="list-style-type: none"> - Prevents misunderstandings. - Aligns all stakeholders on the objectives. 	<ul style="list-style-type: none"> - Misalignment among stakeholders. - Frequent changes in requirements during development. 	[20]
Design Phase	Provides detailed input for architectural and	<ul style="list-style-type: none"> - Efficient and accurate design. - Focus on 	<ul style="list-style-type: none"> - Incomplete requirements cause design flaws. 	[21]

	system design decisions.	scalability and performance.	- Redesign efforts increase costs.	
Development Efficiency	Guides developers with clear instructions on features and functionality.	- Accelerated development process. - Minimal backtracking.	- Misinterpretations lead to rework. - Increased development time and frustration.	[22]
Testing and Validation	Defines test cases and acceptance criteria.	- Easier to verify system behavior. - Thorough validation ensures quality.	- Testing gaps due to undefined or ambiguous requirements. - Higher defect rates.	[23]
Security Implementation	Identifies security requirements early, such as encryption, access control, and data protection.	- Proactive threat mitigation. - Fewer vulnerabilities post-deployment.	- Overlooked security concerns. - Increased risks of data breaches or compliance violations.	[22]
Stakeholder Communication	Provides a reference document for collaboration between developers, clients, and testers.	- Smooth communication. - Fewer misunderstandings.	- Miscommunication leads to unmet expectations. - Stakeholder dissatisfaction.	[21]
Budget Management	Helps in accurate estimation of costs based on detailed requirements.	- Optimized resource allocation. - Reduced budget overruns.	- Underestimated costs. - Additional expenses due to scope creep and rework.	[24]
Product Quality	Ensures all requirements, including usability and performance, are	- High-quality, user-centric product. - Meets or exceeds expectations.	- Poor quality product. - Missed performance or usability targets.	[19]

	met.			
Change Management	Facilitates handling changes with traceable and well-documented requirements.	- Controlled and predictable changes. - Minimal disruption to the development process.	- Difficulty managing changes. - Increased likelihood of scope creep and project delays.	[25]
Post-Deployment Maintenance	Documents requirements for future updates or enhancements.	- Easier maintenance. - Faster resolution of issues.	- Ambiguous or missing documentation hinders updates. - Longer resolution times.	[26]

In table 3, key security parameters to include in a Software Requirement Specification (SRS) to ensure comprehensive security coverage:

Table 3: Security parameters

Security Parameter	Description	Source
Authentication	Ensure secure user identification methods (e.g., username/password, biometric authentication, MFA).	[9]
Authorization	Role-based access control (RBAC) to restrict access to resources based on user roles.	[10]
Encryption	Use strong encryption (e.g., AES-256) for data in transit and at rest.	[11]
Secure Communication	Enforce secure communication protocols like HTTPS, SSL/TLS for data exchange.	[12]
Input Validation	Validate all user inputs to prevent injection attacks, such as SQL injection and cross-site scripting (XSS).	[9]
Output Encoding	Encode output to prevent security vulnerabilities like XSS.	[9]
Data Integrity	Ensure mechanisms for verifying that data is not altered during	[10]

	transmission or storage (e.g., digital signatures).	
Logging and Monitoring	Implement secure logging to record system activities, with monitoring for anomalies.	[11]
Security Auditing	Define auditing requirements to ensure compliance and detect unauthorized activities.	[12]
Session Management	Enforce secure session handling, including session timeouts and secure cookie attributes.	[9]
Access Control	Define fine-grained access permissions for all resources.	[13]
Secure Configuration	Include requirements for secure default configurations for all software and systems.	[10]
Vulnerability Management	Outline processes for identifying and patching vulnerabilities.	[11]
Threat Modeling	Incorporate requirements to conduct threat modeling during the design phase.	[12]
Third-Party Risk Assessment	Specify security expectations for third-party components or integrations.	[14]
Backup and Recovery	Define requirements for secure and regular backups and disaster recovery plans.	[12]
Secure API Design	Ensure APIs use secure authentication and validation practices.	[11]
Privacy Compliance	Address compliance with privacy regulations (e.g., GDPR, CCPA) to protect user data.	[16]
Incident Response	Include requirements for an incident response plan, detailing steps for detecting, mitigating, and reporting breaches.	[13]

Security Testing	Define requirements for penetration testing, vulnerability scanning, and code analysis.	[16]
------------------	---	------

In table 4, key quality parameters for a Software Requirement Specification (SRS) to ensure it meets industry standards and supports effective software development:

Table 4: Key Quality Parameters

Quality Parameter	Description	
Completeness	All functional, non-functional, and security requirements are specified without leaving gaps.	[1]
Clarity	Requirements are unambiguous, precise, and understandable by all stakeholders.	[2]
Consistency	No conflicting or contradictory requirements; aligns with project goals and standards.	[2]
Feasibility	Requirements are realistic and achievable within the project's technical and resource constraints.	[2]
Traceability	Each requirement is traceable to its origin (e.g., stakeholder needs) and linked to design and testing phases.	[2]
Verifiability	Requirements are stated in a way that can be tested or verified during the validation phase.	[3]
Modifiability	The SRS is easy to update as changes occur, without affecting its structure or coherence.	[4]
Prioritization	Requirements are ranked by importance and urgency to guide implementation focus.	[4]
Relevance	All requirements align directly with the system's purpose and stakeholder needs.	[2]
Scalability	The SRS addresses how the system will handle increased workload or scale over time.	[2]
Security	Includes well-defined security measures for data protection, access control, and threat mitigation.	[2]
Performance	Specifies performance requirements such as response time, throughput, and latency.	[3]
Usability	Defines user interface and experience requirements to ensure ease of use for the target audience.	[4]
Maintainability	Outlines requirements for ease of system updates, debugging,	[4]

	and code refactoring.	
Portability	Addresses requirements for system deployment across different platforms, operating systems, or environments.	[2]
Accuracy	Ensures requirements accurately reflect stakeholder expectations and system goals.	[2]
Testability	Includes criteria for creating test cases to validate each requirement.	[2]
Compliance	Aligns with industry standards, legal regulations, and organizational policies.	[3]
Structured Format	Follows a logical and standardized structure to improve readability and comprehension.	[4]
Innovation and Flexibility	Accommodates future innovations and changes in technology or stakeholder needs.	[4]

Software Requirement Specifications Using Intelligent Technical

It is noteworthy to note that there are a great number of scholars who have written studies regarding the classification of detected needs into functional and non-functional requirements. Numerous research studies have been carried out in order to examine the degree to which the AI and ML methodologies are applicable in this particular setting. In 2013, Ramadhani et al. [22] suggested an automated system to find non-functional requirements from sentence-based classification algorithms that are needed for FSKNN. This system would use semantic factors like the development term by hyper name and measure the semantic association between the term and each quality aspect class. Ramadhani and his team designed this system to pinpoint non-functional requirements. The system was designed with reference to ISO/IEC 9126. The result of their research is that the semantic-FSKNN method can reduce the multiplication loss or error rate by 21.9% and also increase the accuracy value by 43.7%; the accuracy value is also 73.9% compared to the FSKNN method without adding semantic factors to it. The researchers tested whether adding semantic elements to the FSKNN method will improve the Hamming loss performance by the amount mentioned above. During the year 2017, Kurtanović and colleagues [23] used over- and oversampling techniques in order to address the issue of unbalanced classes within a dataset. In addition, they checked the accuracy, recall, and F1 metrics of the classifiers in a set of experiments using the Support Vector Machine classifier process. For the purpose of automatically recognizing FRs and NFRs, it achieves an accuracy and recall of up to about 92%. It got the highest levels of precision and recall for security and performance NFRs, with up to 92% accuracy and around 90% recall. This allowed it to find specific NFRs with the highest levels of accuracy. Abad, S. et al. 2017 [24] talked about how to use machine learning algorithms like Latent Dirichlet allocation (LDA), Biodegradable Temporizing Matrix (BTM), Hierarchical, K-means, Hybrid, and Binarized Naïve Bayes (BNB) to make it easier for requirements to be automatically put into two groups: FR and NFR. With regard to the subclassification of NFRs, BNB did the best. While the BTM algorithm performs better than

the LDA algorithm in general, it is not very effective when it comes to subclassifying NFRs. Using this approach, we perform a standardization and normalization of criteria prior to applying classification algorithms. For the research, we extracted 625 requirements from the OpenScience tera-PROMISE repository. Preprocessing significantly enhances the performance of both FR/NFR classification and NFR subclassification. In 2020, S. Tiun et al. [25] introduced software classification using Word2vec and fastText methods to perform text analytics to gain intuition or knowledge from the crowd's feedback. They concluded that "fastText" was the best model for the FR and NFR classifications. Because FastText did better than Word2Vec when using a deep learning classifier, we can say that a deep learning classifier does not always do better than a linear classifier when it comes to text classification problems. For binary text classification with a very short document length and minimal vocabulary, FastText can do a better job. They suggested that if one prefers to use traditional features and classifiers in classification similar to FR and NFR, one should consider using TFIDF with NB as their model, as the model had the highest F1 score of 92.8%. In the same year, Canedo et al. [26] demonstrated that they could manually classify user requirements into two categories, "functional requirements and nonfunctional requirements," by combining two text routing techniques with four machine learning algorithms. The NFR has eleven types (subcategories of non-functional requirements) and twelve types of FR, plus subcategories of non-functional requirements. The researcher found that the combination of TF-IDF and LR had the best performance metrics for binary classification, NFR classification, and requirements classification overall, with an F-measure of 91% for binary classification, 74% for 11-detail classification, and 78% for 12-detail classification.



Figure 1: Show the Factors and their relation [12, 13, 14, 15]

Requirement Sentence Quality (RSQ) and Requirement Document Quality (RDQ)

Requirement Sentence Quality (RSQ) and Requirement Document Quality (RDQ) are two key concepts for ensuring the quality of software requirements. Both are crucial for project success but differ in their focus and scope.

Table 5: Differences Between RSQ and RDQ [4, 6, 8, 26, 28, 29, 30]

Aspect	Requirement Sentence Quality (RSQ)	Requirement Document Quality (RDQ)
Focus Area	Quality of individual sentences.	Quality of the overall document.
Level of Focus	Sentence-level clarity and precision.	Document-level structure and organization.
Objective	Eliminate ambiguity in individual requirements.	Make the document complete, consistent, and easy to navigate.
Consistency	Ensures consistency in terminology	Ensures consistency across the document's sections and

	and grammar at the sentence level.	requirements.
Testability	Makes individual sentences measurable and testable.	Facilitates validation and testing at the document level.
Scope	Focuses on linguistic quality and clarity of single sentences.	Ensures the document addresses all requirements comprehensively.
Traceability	Not applicable.	Maps requirements to their sources and test cases.
Adaptability	Changes in one sentence have minimal impact.	Document must account for change management and trace changes.
User-Focus	Ensures individual sentences are understandable by stakeholders.	Ensures the document addresses all user needs in a structured way.
Error Handling	Focuses on identifying sentence-level errors.	Focuses on addressing inconsistencies or gaps in the entire document.

Secure SRS

Security requirement elicitation

It is the process of researching and uncovering the system's needs from users, consumers, and other stakeholders [13]. Security requirement elicitation is a method that is becoming more popular. This stage is used to pull up effective as well as efficient requirements linked to the security of a system from the statement of the problem (SOP). This stage begins with a statement about the issue. The stages that are included in this are as follows: The process involves identifying stakeholders, assigning roles and responsibilities to each stakeholder, identifying security requirements, identifying assets, and estimating potential threats. The identification of vulnerabilities in functional and non-functional needs and quality characteristics must be done during this phase in order to prevent possible attacks at a later stage of the product's development. This phase is also responsible for defining user expectations for a new or changed product. This phase encompasses the following procedures: conflict resolution, completeness checks, requirements inspection, and review activities. The proposed secure SRS model's security requirement analysis phase should take security into account. There are certain common errors.

Establishing Priorities for Security Requirements: The primary objective of this step is to arrange the security needs in descending order of significance, starting with the most important and working down to the least so. The danger that is there during the development stage will be reduced. The Analytic Hierarchy Process (AHP) is a good way to figure out which of several options is most important when it comes to security. It can choose an option from a large group of options or give options more weight based on how bad the situation. This theory is a relative measuring theory that uses absolute scales to evaluate both physical and intangible criteria. The judgment of those with knowledge and experience in the field

forms its foundation. The decisions that are made are either based on a single number that represents the greatest possible result or on a vector of priorities that provides an ordering of the many outcomes that are conceivable. This process encompasses the following stages: This includes the evaluation of threats, the rating of assets, the prioritizing of threats, the prioritization of security requirements, and the calculation of risk value.

In the realm of the term "**security requirement management**" is used in information security. The term "security requirement management" refers to the process of gathering requirements, conducting analysis on them, ranking them in order of severity, reaching a consensus on their definition, managing changes, and effectively communicating with the relevant stakeholders [13, 14]. This process incorporates the following procedures: Traceability of security needs, viewpoint identification, and both functional and non-functional requirements are all important considerations. It is possible to ensure the safety of the suggested SRS model by including defence mechanisms at this developmental stage. The model we made solves these problems by using the Confidentiality, Integrity, and Availability (CIA) trinity. This is built on by ideas like authentication, authorization, and accounting, as well as scalability and non-repudiation factors.

Critical Observation

- The SLR effectively categorizes the diverse applications of ML techniques for securing SRS, such as ambiguity detection, risk assessment, compliance verification, and requirement classification.
- The inclusion of multiple ML models, such as BERT, SVM, Decision Trees, and Transformers, highlights the growing sophistication of tools applied to SRS-related challenges.
- NLP-based approaches dominate the field, underscoring their relevance in handling textual data inherent in SRS.
- The review rightly prioritizes security as a critical aspect of SRS. By addressing ambiguities, inconsistencies, and vulnerabilities in requirements, ML contributes significantly to preempting security flaws in the software development lifecycle (SDLC).
- The review highlights the lack of standardized, annotated datasets for SRS-related tasks. This limits the reproducibility and benchmarking of ML techniques.
- Few studies address how ML models can be scaled to handle large, complex SRS documents in real-world scenarios or integrated into Agile and DevOps workflows.
- The review notes that most studies rely on controlled environments or synthetic datasets. Real-world validation of ML models in dynamic industry settings is largely missing.
- There is limited research on integrating ML-based SRS solutions with industry tools like JIRA, Confluence, or other SDLC tools.

References

1. Ramadhani, D. Rochimah, S. Yuhana, U. "Classification of Non-Functional Requirements Using Semantic-FSKNN Based ISO/IEC 9126" Dian Nuswantoro University Imam Bonjol, Semarang 50131, Institut Teknologi Sepuluh Nopember, Surabaya 60111, Indonesia, 2013.

2. Kurtanović, Z. Maalej, W. “Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning” Lisbon, Portugal, 2017.
3. Abad, S. Karras, O. Ghazi, P. Glinz, M. Ruhe, G. Schneider, K. “What Works Better? A Study of Classifying Requirements” University of Calgary, Calgary, Canada, Leibniz Universität Hannover, Hannover, Germany, University of Zurich, Zurich, Switzerland, arXiv:1707.02358v1 [cs.SE] 7 Jul 2017.
4. S Tiun, U A Mokhtar, S H Bakar, S Saad, “Classification of functional and non-functional Citation of this Article: requirement in software requirement using Word2vec and fast Text” Universiti Kebangsaan Malaysia, 43600 Bangi, Selangor, Malaysia, 2020.
5. Canedo, E. Mendes, B. “Software Requirements Classification Using Machine Learning Algorithms” University of Brasília (UnB), P.O. Box 4466, Brasília 70910-900, Brazil, 2020.
6. Requirements Managements, [Online]. Available: [https://en.wikipedia.org/wiki/Requirements managements](https://en.wikipedia.org/wiki/Requirements_managements).
7. P. Lokhande, B. Meshram, “Analytic Hierarchy Process (AHP) to Find Most Probable Web Attack on an E-Commerce Site”, ICTCS, ACM, March 2016.
8. Thomas L. Satty, “The Analytic Network Process,” University of Pittsburgh
9. Amelia, O. (2024). AI-Powered Software Testing Automation: Machine Learning Approaches for Ensuring Quality in Distributed Systems. *ResearchGate*.
10. Patil, V., & Jadhav, P. (2024). Integrating AI-Powered Chatbots and PDF Processing for Enhanced Document Management: A Full-Stack AI SaaS Approach. *ResearchGate*.
11. Hudaib, A., Salem, H., & Al-Tarawneh, K. (2024). Generating database schema from requirement specification based on natural language processing and large language model. *CRM*, 12(7), 23–40.
12. Ontiveros, J. A. (2024). Natural Language Processing for Automated SysML Diagram Generation. *ProQuest Dissertations*, 24(4), 12–25. PDF Link
13. Alqurashi, S. S., Ray, I., & Malaiya, Y. (2024). Towards automated security and privacy policies specification and analysis. *Mountain Scholar*, 15(2), 30–45
14. Mattioli, J., Awadid, A., & Sohier, H. (2024). Trustworthy ML Assessment methodology. *HAL Archives*, 8(1), 41–60.
15. Kolhe, K. P., & Jima, B. A. (2025). Classifying software security requirements into confidentiality, integrity, and availability using machine learning approaches. *PeerJ Computer Science*, 6(2), e2554.
16. Mishrif, A., & Al Qamashoui, A. (2024). Breaking Barriers: An In-Depth Examination of Entrepreneurial Policies for People with Disabilities in Oman. *Springer*, 18(3), 112–129.

17. Guendouzi, B. S., & Ouchani, S. (2025). Ensuring the federation correctness: Formal verification of Federated Learning in industrial cyber-physical systems. *Future Generation Computer Systems*, 137(4), 101–120.
18. Liu, J., Xiao, W., & Cheng, H. (2024). Graph neural network-based time estimator for SAT solver. *Springer*, 42(7), 87–102.
19. D. Pandey, U. Suman, and A. K. Ramani, “An Effective Requirement Engineering Process Model for Software Development and Requirements Management,” in Proceedings of the International Conference on Advances in Recent Technologies in Communication and Computing, 2010, pp. 287–291.
20. S. O. Mokhtar, R. Nordin, Z. A. Aziz and R. M. Rawi, “Issues and challenges of requirement review in the industry.” *Indian Journal of Science and Technology*, vol. 10, no. 3, pp. 1-5, Jan. 2017.
21. F. Fabbrini, M. Fusani, S. Gnesi and G. Lami, “Quality evaluation of software requirement specifications,” in Proceedings of the Software and Internet Quality Week 2000 Conference, 2000, pp. 1–18.
22. V. Rastogi, “Software Development Life Cycle Models-Comparison, Consequences,” *International Journal of Computer Science and Information Security*, vol. 6, no. 1, pp. 168–172, 2015.
23. Jurca G, Hellmann TD, and Maurer F (2014). Integrating agile and user-centered design: A systematic mapping and review of evaluation and validation studies of agile-UX. In the Agile Conference, IEEE, Kissimmee, USA: 24-32. <https://doi.org/10.1109/AGILE.2014.17>
24. Ågren SM, Knauss E, Heldal R, Pelliccione P, Malmqvist G, Bodén J (2019) The impact of requirements on systems development speed: a multiple-case study in automotive. *Requirements Eng* 24(3):315–340
25. Alsaqaf W, Daneva M, Wieringa R (2019) Quality requirements challenges in the context of large-scale distributed agile: an empirical study. *InfSoftwTechnol* 110:39–55
26. Behutiye W, Karhapää P, Costal D, Oivo M, Franch X (2017) Non-functional requirements documentation in agile software development: challenges and solution proposal. In: Proceedings of the 18th International Conference on Product-Focused Software Process Improvement (PROFES 2017), pp 515–522
27. Franch X, Mendez D, Vogelsang A, Heldal R, Knauss E, Oriol M, Travassos G, Carver JC, Zimmermann T (2022) How do practitioners perceive the relevance of requirements engineering research? *Trans SoftwEng* 48(6):1947–1964
28. Garousi V, Borg M, Oivo M (2020) Practical relevance of software engineering research: synthesizing the community’s voice. *EmpirSoftwEng* 25:1687–1754
29. Palomares C, Franch X, Quer C, Chatzipetrou P, López L, Gorschek T (2021) The state-of-practice in requirements elicitation: an extended interview study at 12 companies. *Requirements Eng* 26(2):273–299

30. Wagner S, Mendez Fernandez D, Kalinowski M, Felderer M, Mafra P, Vetrò A, Conte T, Christiansson M-T, Greer D, Lassenius C, Männistö T, Nayebi M, Oivo M, Penzenstadler B, Pfahl D, Prikladnicki R, Ruhe G, Schekelmann A, Sen S, Spinola R, de la Vara JL, Tuzcu A, Wieringa R, Winkler D (2019) Status quo in requirements engineering: a theory and a global family of surveys. *Trans SoftwEngMethodol* 28(2):9