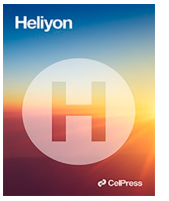
Heliyon 10 (2024) e33495

Contents lists available at ScienceDirect

Heliyon

journal homepage: www.cell.com/heliyon



Research article

A neuro-fuzzy security risk assessment system for software development life cycle

Olayinka Olufunmilayo Olusanya [a], Rasheed Gbenga Jimoh [b], Sanjay Misra [c,d,*], Joseph Bamidele Awotunde [b]

[a] Department of Computer Science, Tai Solarin University of Education, Ijagun, Ogun State, Nigeria
[b] Department of Computer Science, Faculty of Communication and Information Sciences, University of Ilorin, 240003, Kwara State, Nigeria
[c] Department of Computer Science and Communication, Østfold University College, Norway
[d] Department of Applied Data Science, Institute for Energy Technology, Halden, Norway

ARTICLE INFO

Keywords:
Security risk assessment
Software development life cycle
Software development
Neuro-fuzzy modeling

ABSTRACT

This study aims to protect software development by creating a Software Risk Assessment (SRA) model for each phase of the Software Development Life Cycle (SDLC) using an Adaptive Neuro-Fuzzy Inference System (ANFIS) model. Software developers discovered and validated the risk variables affecting each SDLC phase, following which relevant data about risk factors and associated SRA for each SDLC phase were collected. To create the SRA model for SDLC phases, risk factors were used as inputs, and SRA was used as an output. The formulated model was simulated using 70 % and 80 % of the data for training, while 30 % and 20 % were used for testing the model. The performance of the SRA models using the test datasets was evaluated based on accuracy. According to the study findings, many risk variables were discovered and confirmed for the requirement, design, implementation, integration, and operation phases of SDLC 11, 8, 9, 4, and 6, respectively. The SRA model was formulated using the risk factors using 2048, 256, 512, 16, and 64 inference rules for the requirement, design, implementation, integration, and operation phases, respectively. The study concluded that using the SRA model to assess security risk at each SDLC phase provided a secured software development process.

1. Introduction

SDLC is a process or structure imposed on the development of a software system within a software organization [1]. At every stage of the software development process, the SDLC outlines the duties that must be carried out, thus the model is used in the software industry. When this is done well, the SDLC creates high-quality software that fulfills customer expectations and is finished on schedule and under budget [2,3]. The SDLC process is also seen as a series of steps that system designers and developers must do to produce quality software within a set amount of time [4]. There are numerous SDLC approaches and models; however, regardless of the choice of model, different security risks are associated with each of these SDLC phases.

Managing and integrating security into the SDLC is one of the most important problems that businesses face [5]. The development of secure software applications can be aided by a collection of software security standards, guidelines, practices, and certifications

* Corresponding author. Department of Computer Science and Communication, Østfold University College, Norway.
E-mail addresses: yinka_olusanya@yahoo.com (O.O. Olusanya), jimoh_rasheed@unilorin.edu.ng (R.G. Jimoh), sanjay.misra@hiof.no, sanjay.misra@ife.no (S. Misra), awotunde.jb@unilorin.edu.ng (J.B. Awotunde).

https://doi.org/10.1016/j.heliyon.2024.e33495
Received 16 May 2023; Received in revised form 7 June 2024; Accepted 21 June 2024
Available online 22 June 2024

[4–8]. Still, only a small number of scenarios have been published, even though it is well-acknowledged how important it is to offer scenarios that span the whole secure software development life cycle (SDLC) [8–10]. To address the technological and psychological aspects of the issue, it is becoming more and more crucial to comprehend secure software engineering (SSE) approaches [9]. SSE, which encompasses secure SDLC procedures and Secure software development (SSD) techniques. This is the process of developing, constructing, and testing software to make it secure [4]. The following four procedures must be followed to create secure software: Security related to software requirements, design, implementation, and testing [9–11]. The processes aim to enhance software security by utilizing threat modeling techniques, implementing stricter security standards, and adhering to best practices during coding, code review, and testing. To guarantee that software products are secure, this procedure needs to be updated frequently; hence, research highlighting the patterns in methods, notations, tools, and approaches is required [9].

Traditionally, security testing in software development occurs during the last phase of the SDLC; however, identified bugs cause little or no problem to software functionality but, in most cases, to the security of the software [12,13]. As a result of this, privacy must be included in the design procedure from the start. It must keep improving the safety of deployed software to software throughout the design phase implementation [6,14]. Using the software risk management procedure to build software allows organizations to reconcile system needs for protecting corporate data and resources with the expense of protection measures and mitigation measures across the SDLC [15–17].

The SDLC process is exposed to hazards from the early stages of the project until the software product is finally accepted by its target consumers [18,19]. This is because each stage of the Software lifecycle is vulnerable to different types of risks that might prevent the development cycle from being executed effectively [20,21]. To adequately address these risks, a better comprehension of the difficulties and dangers connected with the software engineering process is essential to be identified. Software security risk factors can be defined as those items that might hinder a software project from achieving its expected outcomes or cause it to fail [22,23]. If these risk factors are ignored or not mitigated, they can cause a serious threat to the success of a software project [24–26]. How well risks are managed throughout the SDLC has been linked to the production of secure, elevated, error-free applications within budgetary and financial projections [27,28]. Each phase of the SDLC is linked to a certain type of software security risk. Few models or tools are currently available that can help developers conduct SDLC security risk assessments. The existing studies have revealed various risk factors associated with security risk in SDLC as a whole rather and failed to check for each stage of the software development using the SDLC process. Several risk factors associated with each phase of the SDLC in the context of software developers were identified based on a risk factor checklist taxonomy for each SDLC phase as proposed by authors in Refs. [1,2].

A fundamental idea in fuzzy logic is the incorporation of structured human knowledge into executable algorithms to create fuzzy models. It was created to eliminate ambiguity and imprecision and to enhance the robustness, tractability, and affordability of solutions to real-world issues [29]. A given element can belong to more than one fuzzy set simultaneously, which makes it based on graded concepts and membership elasticity [30]. Membership functions are triangular, trapezoidal, and Gaussian, defining how input space was mapped to an interval [0, 1]. Adaptive neuro-fuzzy inference systems are feed-forward networks with multiple layers combined with a fuzzy-based model [31]. In short, they are systems that combine the characteristics of neural network architecture and a fuzzy logic system. The security risk at each stage of SDLC can be assessed by software developers using the Web-based Information Expert System proposed in this study, thus making it easily available and accessible to the stakeholders. The adaptive neuro-fuzzy model used in this study was developed for security risk assessment using risk variables found by software developers.

Previously, security considerations were included as an afterthought to software development, only taking into account software security threats at a later stage [4]. This raises the possibility of adding fresh security risks at different phases of the SDLC process. Each step of the SDLC is known to be connected to various types of software security risk, and there are no established models or tools available to help developers with security risk assessment in each stage throughout the SDLC [5]. The existing models and methods in recent years can only assist software developers and project managers in carrying out security risk assessment once during the SDLC process. Most of the existing models have one limitation or the other ranging from lack of update facility to failure to identify the risk factors associated with software risks at each phase of SDLC [9]. Hence, in studies where security risk assessment was performed using risk factors, they were only limited to the whole SDLC as a single process.

Therefore, this study will extend the previous models by developing an ANFIS model for assessing the security risk associated with each stage of the SDLC process. Each stage of the SDLC process is vulnerable to security risks from the start of the project until the time of acceptance of the software product. The SDLC is subject to various threats at each phase, which can potentially hinder the successful completion of the development process. This new model involved Nigerian software developers in risk validation at all stages of SDLC by adapting the risk factors available at different stages of SDLC on the risk factors checklist taxonomy as presented in the literature. The model was implemented as a web-based information system. The following are the main contributions of the proposed model:

(i) To give a qualitative investigation of the practices used in actual software security.
(ii) To determine and assess the risk elements required for assessing security risks at each phase of software development.
(iii) To formulate ANFIS model for SRA at each SDLC phase using risk factors in (i);
(iv) To simulate and evaluate the performance of the model; and
(v) To implement a web-based prototype system of the model.

The ANFIS Security Risk Assessment System tailored for the SDLC can contribute to more accurate, adaptive, and efficient security practices within software development companies. It aligns well with the dynamic nature of the field and helps in building more secure software applications from the early stages of development. By integrating the ANFIS-based model into the SDLC, security risks can be identified at both stages and in an early stage. This allows development teams to address vulnerabilities during the planning and design

phases, reducing the cost and effort required to fix issues later in the development process or after the software is deployed. The proposed model can be customized and trained based on the specific characteristics and requirements of a software project. This tailoring allows the system to provide more relevant and accurate risk assessments that align with the unique aspects of each software development endeavor.

The article is divided into six parts. The following section provides a literature review. Material and methods are provided in section 3. Implementation, results, discussion, and conclusion drawn are in sections 4 to 6, respectively.

2. Related works

A security metric assesses software security performance, goals, and objectives and offers data that aids in software security assessment [32,33]. With the advent of technology, software faces numerous challenges and risks to the data and privacy of clients. The majority of these dangers arise due to security being overlooked as a characteristic of software development [14,34,35]. The essential security attributes of software, namely confidentiality, integrity, and availability, are greatly improved by security metrics gathered during the development process [36]. As a result, security should be considered throughout the software development lifecycle, so software security metrics are used. In this direction, Authors in Refs. [32,33] examine the several security metrics that are contemplated throughout the numerous phases of the lifecycle's evolution to provide practitioners, managers, and researchers with substantial knowledge for further security evaluation.

The evolution of program cyber threats has been influenced by the advancement of attackers, the spread of new technologies, and the use of complex platforms. Code security and data confidentiality must be integrated with both design and development practices to comply with such scenarios. The researchers developed a Privately Oriented Software Development (POSD), a software design technique that extends standard development procedures by merging software privacy and protection management activities. The methodology's five key components are Confidentiality by Development, Security Design Strategies, Security Structure, Security flaws, and Circumstances. The approach may be used to develop new software products or re-engineer existing ones in two ways: forward and backward. The findings show that POSD can detect software problems, discover the remediation processes required to address them in the source code, and construct the target architecture for driving privacy-oriented interaction effects.

With the rise of threats, it's more important than ever to address security in the SDLC [37,38]. Vulnerabilities exist despite attempts to develop a secure SDLC and literature recommending methods and ways to help in the process of detecting and removing weaknesses [39,40], security flaws continue to be discovered and eliminated. Developers are sometimes referred to be "the poorest link in the chain" and are held accountable for security issues [41,42]. Nevertheless, requiring engineers to continually expand their security measures is impractical and unlikely to provide outcomes [42,43]. The study highlights the lack of research on human aspects of software security, with programmers often viewed as the weakest link, a trend that persists before useable security research became significant. While engineers are more technically skilled than average end-users, they should not be considered security specialists [43]. When it comes to security responsibilities, they require assistance, such as authentication for developers [44] or computer languages that avoid security errors [45,46]. To that aim, Acar et al. [43] proposed a research agenda to quickly identify programmers' perspectives and security expertise, examine the usefulness of current security development tools, and provide tools and methodologies to help developers design secure applications. We'll now go over some recent studies on human aspects of software security.

An artificial neural network was used by Gandhi et al. (2014) [26] to create risk management software. The study compiled ten risk criteria from various specialists that are connected to evaluating security risk throughout the SDLC stages. The study made use of 10 risk factors to gather pertinent information for evaluating the security risk connected with software development. The study was only able to identify 10 risk variables, which could only be utilized to assess safety threats once the SDLC procedure was complete.

Babaie [33] developed a novel expert fuzzy-logic tool that can be used by project decision-makers to evaluate the expected risks and help in estimating the risk probability based on the software project's critical success factors. The paper does not mention a specific "Neuro-Fuzzy Security Risk Assessment System" for the SDLC. A neuro-fuzzy technique was presented by Ansari et al. [34] to evaluate the software's reliability and deal with measurement uncertainties for the software's actual parameters. By creating neuro-fuzzy inference systems, utilizing the four ISO/IEC 9126 quality model parameters (fault-tolerantness, recoverability, reliability compliance, and software maturity), and estimating a fuzzy system's parameters by taking advantage of neural network approximation techniques, one can find the parameters of a fuzzy system. Experts and software developers have determined the weights of the input parameters, and MATLAB is used to run the simulations (ANFIS). Simulations verify that the suggested approach yields satisfactory outcomes.

Although quantitative security assessment is recognized as important in secure software development, there is currently no effective model for quantifying internal software security risk in the literature. In order Sivvas et al. [35] present a multilayer security evaluation method (SAM a security-relevant static analysis alarms and program measures) to evaluate the interior security level of software applications using low-level indications. The approach, which is built on criteria and balances and adheres to ISO/IEC 25010 regulations, painstakingly integrates various low-level signs to create a high-level security score showing the internal security level of the researched program. The proposed technique is feasible since it is completely automated and operationally defined as a standalone tool as well as a component of a broader Computer-Aided Software Engineering (CASE) framework. The model's thresholds to boost reliability, it was validated based on a repository of 100 major software packages taken from the Maven Repository. The trials show that the proposed model can evaluate domestic security with sufficient discretion at both the item and category levels of granularity. They also provide early evidence for the model's capacity to be used as a vulnerability indicator. To the best of our knowledge, this is the first fully automated, operationalized, and well-tested security evaluation process in the current literature.

Backes et al. [47] proposed a fuzzy-based symmetrical decision-making approach to assess the security of software during the

Table 1
Summary of software risk management models.

Author(s)	Area	Model-based	Software Platform	Strength	Weakness
Gandhi et al. [27]	Risk management	Artificial neural network	MATLAB	Involvement of experts in risk identification	Ten risk factors
Ibraigheeth & Fadzli [33]	CHAOS	Neuro-Fuzzy Model	MATLAB	Developed a novel expert fuzzy-logic tool	The paper does not mention a specific "Neuro-Fuzzy Security Risk Assessment System" for the SDLC
Babaie [34]	Generic	neuro-fuzzy	MATLAB	An integrated assessment paradigm for software reliability evaluation	Only for risk factors in software reliability test
Agrawal et al. [47]		fuzzy-based symmetrical decision-making		The fuzzy Analytic Network Process (F-ANP) is utilized to assess the fuzzy-symmetrical approach and criteria weights.	The model was applied to software products and not during the SDLC process
Sharif and Rozan [39]	SME Project Time Management Risk Assessment	Risk Assessment Tool (RAT) Model	Oracle Application Express (Apex) 3.2 and Oracle Database 11g	Project managers can easily identify project time risks using project plan data as inputs	Used to assess and monitor project time management risks.
Khan et al. [41]	Global software development (GSD)	fuzzy analytical hierarchy process (FAHP)	Unknown	Because FAHP is useful in addressing multicriteria decision-making problems, it is innovative in this field.	inappropriate strategy for identifying, enforcing, authenticating, authorizing, and protecting privacy in secure requirements," and "inadequate update of threat models."
Suresh & Dillibabu [49]	NASA 93	ANFIS MCDM	Unknown	combined intuitionistic fuzzy-based TODIM (IF-TODIM) techniques, adaptive neuro-fuzzy inference system-based multi-criteria decision-making (ANFIS MCDM), and fuzzy decision-making trial and evaluation laboratory	Take the risk factors at once throughout the SDLC process
Sharif et al. [50]	Risk Assessment	Review of Existing Model	None	Classifies risk assessment models as risk awareness-based, risk identification-based, risk ranking and decision-taking types	Assessment of security risk of software models developed for small and medium scale projects.
Mustafa et al. [52]	Generic	Probabilistic Inference Model	Unknown	Facility for risk update available. Used PIM for subjective judgment of users	No prioritization of risk, poor risk analysis
Gambo et al. [53]	Requirements Engineering	Rank Based	Delphi 7.0	Provides a means of prioritizing requirements	Limited to the requirement analysis phase of the SDLC
Pooja and Dalwinder [54]	Risk estimation	Neuro-fuzzy	MATLAB	A hybrid approach, Involvement of experts in risk identification	17 risk factors
Batar et al. [55],	Generic	ANFIS	MATLAB	The model used in the rule-based software risk assessment and management approach is reliable and accurate, with a low average testing error and a high accuracy rate.	Take the risk factors at once throughout the SDLC process

development life cycle. The proposed symmetrical assessment in this study will be beneficial for both the designers and developers to categorize and prioritize the security attributes and understand the importance of security tactics during the software development life cycle. Moreover, the risk assessment tool (RAT) developed by Sharif and Rozan [39] is a risk assessment tool for project time. The tool focused on the management aspect of the project risks by systematically identifying, analyzing, and prioritizing the time risk of the project based on project documentation and project tasks. This is without involving the experts in risk identification. These risks created an important gap as involving an expert in risk identification is very important.

In general, investigations in this field have a difficult time finding developers and verifying ecological validity. Developers have a lot on their plates, and they frequently have to adhere to corporate limits on what may be released publicly. Acar et al. [48] created a tool for conducting distributed online programming research with engineers to address some of these issues. Security flaws are "blind spots" in developer decision-making processes, according to Suresh & Dillibabu [49], who found that developers' decision-making processes are primarily focused on functionality and performance. Sharif et al. [50], focusing on software security resources in general, discovered that some safety guidance is out of date and that the majority of materials lack actual examples. They also discovered certain disciplines that were underrepresented, such as program analysis tools.

Witschey et al. [51] proposed a hybrid fuzzy-based ML-based mechanism for risk assessment in software projects, which includes an adaptive neuro-fuzzy inference system-based multi-criteria decision-making approach. The findings show that the proposed integrated fuzzy approaches can exhibit better and more accurate performance in the assessment of software project risks compared to other existing approaches. On the other hand, Mustafa et al. [52] developed a tool called the risks assessment visualization tool (RAVT) which used a probability inference model as a way to represent the subjective judgment of experienced software project managers. The tool provided the possibility of adding, updating, and deleting new risk questions but prioritization of risk was not available. A neuro-fuzzy system is proposed for security risk assessment in the SDLC [41]. The system is used to assess the reliability of software by considering parameters such as software maturity, fault tolerance, recoverability, and reliability compliance. Gambo et al. [53] looked into the factors that influence the adoption of security tools, such as tool quality and the personalities and experiences of the developers. They discovered that more experienced developers are more likely to use security tools and that tool complexity is a barrier to adoption.

Pooja and Dalwinder [54] developed a fuzzy expert system coded using MATLAB that only calculated the total risk of the project but failed to identify any risk factor. Also, no provision of facilities for adding, deleting, updating, and risk prioritization of new risks. Another tool called Project Risk Assessment Decision Support System (PRADSS) was made available for software projects based on a decision support system that made risk prioritization possible but did not support the identification of risk factors associated with a security assessment.

Batar et al. [55] proposed the development of a rule-based software risk assessment and management method using a fuzzy inference system integrated with an ML-based algorithm. A new and original rule set, which could be used and carried out effectively in software risk assessment and management, has been designed and developed based on the implementation of a fuzzy approach technique integrated with an ML-based algorithm. The study identified 17 risk indicators that are linked to expert appraisals of safeguarding threats using the software risk estimating method developed by Srisaeng et al. [56]. The study gathered relevant data and based on the 17 risk criteria involved in determining security risk after the SDLC process, an ANFIS model was developed. The study was limited to the development of ANFIS models for the identification of risk factors in each stage of the SDLC process, and 17 risk factors were established at the most relevant factors in each stage.

Ghiasi et al. [57] prioritized user requirements based on initial needs, which several users identified. The study conducted an interview with various categories of system users for the initial identification of a broad category of requirements. The requirements were later assessed based on a Likert scale rating of the agreement by all system users, following which a cluster-based technique was adopted for facilitating requirement prioritization. The study was limited to prioritizing user requirements for a specific software project instead of all SDLC phases for generic software projects. Table 1 shows the summary of related work on software risk management models.

Quantitative security assessment is a crucial technique in software development security, but there is a lack of an efficient model in literature for estimating internal software security risk. Hence, vulnerabilities remain amid efforts to safeguard the SDLC by several previous studies that offer tools and techniques. Therefore, this study formulates a neuro-fuzzy-based security risk assessment model based on relevant data collected from software development companies regarding the risk factors associated with each SDLC phase. The study discovered in each stage of the SDLC process many risk factors, thus the safety risk was assessed. The security dangers in each stage of the SDLC process in the context of software development are related to each stage finding these risk variables. To explain the binary labels of the detected risk variables based on multiple distinct intervals, the study additionally developed triangular membership functions. The work was restricted to creating a fuzzy inference system of the Mamdani type for evaluating security risk during SDLC phases.

To measure the risk factors state of the SDLC at every stage of the development process, it is becoming more and more crucial to build tools for assessing and managing software risks. Every firm needs to concentrate on risk management for its software initiatives. To prevent possible project failure, risk mitigation techniques should be devised when risk factors are reported. Even though taking software risk into account has become crucial, project decision-makers have access to just a small number of developed tools that may be used to assess and mitigate potential risks. Hence, veracious models have been proposed to solve these risk factors in the SDLC process but failed to assess these risk factors at each stage of the SDLC during software development. Therefore, this study proposed an ANFIS model that can identify SDLC risk factors at each stage of software development.

3. Material and methods

This study used an ANFIS model, which combined the advantages of fuzzy logic and artificial neural network architecture. The benefits of using the fuzzy model include its white box expert system that uses the aggregate of rules to solve a complex problem and defines the underlying relationship that exists between the input and output variables representing the rule base. However, the disadvantages include its non-adaptive model behavior and its high dependence on expert knowledge. The advantage of the artificial neural network included the adaptive network behavior since training data (historical data) was used to adjust the structure of the model by updating the weights. At the same time, the disadvantage is its black-boxed model behavior since the structure of the final model may be established.

An important fact is that incorporating risk assessment activities into the SDLC process is a significant improvement in producing secured high-quality software. From the literature, previous fuzzy and neural network models developed for security risk assessment during SDLC have one limitation or the other. This research developed a new model using the adaptive neuro-fuzzy inference system (ANFIS) approach. Developers in Nigeria identified and validated risk factors for each phase of the Software Development Life Cycle (SDLC), and provide real-life data on risk factors and associated SRA for each phase of the SDLC process. The SRA model was formulated using an ANFIS technique for each phase of SDLC such that the risk factors were inputs, while the SRA output was classified as No, Low, Moderate, and High risk. The formulated model was simulated using 70 % and 80 % of the real-life data for training, while 30 % and 20 % for testing the model for each SDLC phase. The performance of the models using the test datasets was evaluated based on accuracy.

The ANFIS model can be taught, allowing for the fuzzy logic model to not rely on expert knowledge. The ANFIS method has the advantage of fusing linguistic and numerical skills. ANFIS also makes use of the ANN's capacity to categorize data and find patterns. Compared to the ANN, the ANFIS model is more transparent to the user and produces fewer memorization mistakes. As a result, the ANFIS has several benefits, including its flexibility, nonlinearity, and potential for quick learning [58].

This method is essentially a rule-based fuzzy logic model, with rules established during the model's training [59]. Data is used to guide the training process. The membership function parameters for ANFIS' fuzzy inference system (FIS) are determined from the training samples. Mamdani and Sugeno are the most widely used fuzzy inference systems. The fundamental distinction difference between Mamdani and Sugeno is that the output membership functions of the Sugeno scheme are either linear or static [60]. The output membership functions of the Mamdani system, on the other hand, can be triangular, Gaussian, and so on. Sugeno-type fuzzy inference systems are more computationally intensive than Mamdani-type fuzzy inference systems, which were chosen in this work [61]. The Mamdani personality relies heavily on expert knowledge. The Sugeno type, on the other hand, is based on real-world facts.

Following the construction of the SRA model, which was simulated using the MATLAB SRA model toolbox, the development process began with the risk factors identified that are related to the security risk assessment of each stage of software development using the SDLC method. The effectiveness of the SRA model was then assessed after it was integrated into the proposed system.

3.1. The validation of risk factors from each stage of the SDLC process

There is the likelihood that Nigerian native software developers may not consider all but a proportion of these risk factors identified by authors in Ref. [1]. To shortlist the necessary but important risk factors, questionnaires were designed to determine the risk factors that are relevant in SDLC from Nigerian-based software developers from the initially identified risk factors by authors in Ref. [1]. Each risk factor was assessed based on a relevance scale of either Yes (if considered) or No (if not considered). The questionnaires were designed using Google Forms®, a value-added service provided by Google® for GMail® users for conducting the survey. The questionnaire used was attached as a supplementary file.

The software developers were selected from the community of Nigerian developers consisting of project managers, system analysts, front-end and back-end programmers, and user interface (UI) and user experience (UX) developers. The risk factors as proposed by Hijazi et al. (2014), were identified initially according to their respective phase of the SDLC. These initially validated risk factors were proposed to the Nigerian developers from which the most relevant risk factors were selected. Table 2 shows a description of the risk factors that were considered in the requirement analysis and definition phase of the SDLC. In all, there were 21 risk factors which were categorized into five different groups of activity consisting of some associated risk factors.

3.2. Identification of SDLC stage risk factors

Initial research on risk factors used the risk factor taxonomy developed by Hijazi et al. [1], and a recent study by Jimoh et al. [2], to

Table 2
Distribution of risk factors associated with SDLC phases.

Phases of SDLC Cycle	Initially Identified	Selected Risk Factors	Percentage Selected (%)
Requirement Analysis/Definition Stage	21.0	11.0	52.4
Design Stage	19.0	8.0	42.1
Implementation/Unit Testing Stage	23.0	9.0	39.1
Integration/System Testing Stage	16.0	4.0	25.0
Operation/Maintenance Stage	14.0	6.0	42.9

identify a number of them. Software engineers from various businesses identified various risk factors that are important to creating software using the SDLC process. The software developers' responses were classified into three categories: prioritized risk factors, which described in detail the threats that all programmers consented to, very few possible risks, which described in detail the risk aspects that some designers consented to, and no prioritized risk factors, which described the risk elements that no programmers agreed on. Lastly, the high-priority risk factors selected by all software developers were judged to be the appropriate risk criteria for evaluating security risk at each SDLC phase. Fig. 1 displays the flowchart of the proposed system using the SRA model for the identification of security risks in each stage of the SDLC process.

Table 2 lists the notable risk factors that were selected from the original list of risk factors published by Hijazi et al. [1], and the risk factors by the authors' in Ref. [2]. According to the table, 11 risk factors (52.38 %) were chosen from a list of 21 risk factors for requirement analysis and definition, 8 risk factors (42.1 %) from a list of 19 risk factors for design, 9 risk factors (39.1 %) from a list of 23 risk factors for implementation and unit testing, and 4 risk factors (25.0 %) from a list of 16 risk factors for integration and system testing. 38 risk variables in total were used to evaluate security risk throughout the SDLC phases.

Fuzzy analysis tables can be a useful tool for conducting a Security Risk Assessment in the SDLC. These tables help to assess the likelihood and impact of various security risks using fuzzy logic, which allows for more nuanced and qualitative analysis. Table 3 shows the fuzzy analysis for the identified risk factors in each stage of the SDLC process.

3.3. Formulation of neuro-fuzzy (ANFIS) security risk assessment model

The proposed model used the identified risk factors at each SDLC stage as inputs in the SRA model. Hence, there were five different SRA models used in the proposed model, one for each stage of the software development using the DLC technique. The security risk assessment level for the stage for which the SRA model was developed was the output. The SRA model constructs risk components based on the level of potential threat at each stage. Considering P is a stage in the system development life cycle, the group of risk factors known as Ri are those connected to the SRA during the SDLC phase. As an outcome, the risk factors associated with each stage of SP have an impact on the SRA at each of those stages. As a result, at each phase, equation (1) may be utilized to express the SRA. SRA may be used to analyze any of the four risks identified (No Risk, Little Risk, Medium Risk, and Extreme Risk), as demonstrated by

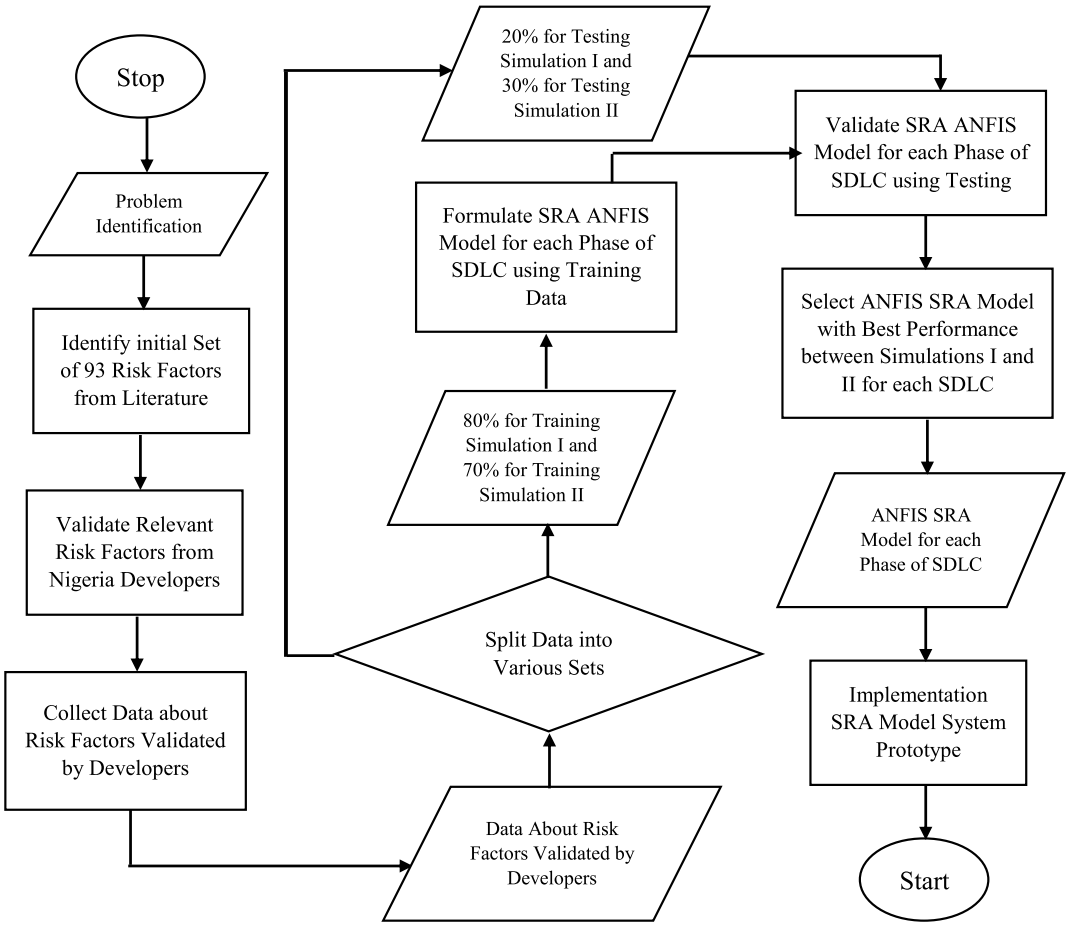


Fig. 1. Research methodology process.

Table 3
The fuzzy analysis tables.

Risk Category	Risk Factor	Likelihood (L)	Impact (I)	Risk Level (L x I)	Risk Rating
Requirement Analysis and Definition	Incomplete or Inadequate Requirements	Low	High	Moderate	Moderate
	Lack of Security Requirements	Moderate	High	High	Moderate
	Unclear Data Handling and Privacy Requirements	High	High	High	High
	Inadequate Stakeholder Involvement	High	High	High	High
	Insufficient Threat Modeling	Moderate	Moderate	Low	Low
	Misaligned Business Goals	Low	Low	Moderate	Low
	Scope Creep	Low	Low	Low	Low
	Third-Party Dependencies	Low	Low	Low	Low
	Insufficient User Input	Low	High	Moderate	Moderate
	Regulatory Compliance Ignorance	Low	Moderate	Low	Low
	Change Management Challenges	Medium	High	High	High
Design Stage	Insecure Architectural Design	Low	Moderate	Low	Low
	Lack of Secure Communication	Medium	High	High	High
	Inadequate Access Control	High	High	High	High
	Insecure Data Storage	Moderate	Moderate	High	Moderate
	Insufficient Error Handling	High	High	High	High
	Injection Vulnerabilities	Low	Low	High	Low
	Insecure File Handling	Moderate	High	High	High
	Insecure Authentication Mechanisms	High	High	High	High
Implementation Stage	Failure to Validate Input	Moderate	High	High	High
	Insecure API Implementations	Moderate	Moderate	Moderate	Moderate
	Lack of Proper Error Handling	High	High	High	High
	Hard-Coded Secrets	High	High	High	High
	Inadequate Session Management	Low	Moderate	Moderate	Moderate
	Insecure File and Data Handling	High	High	High	High
	Code Review Neglect	High	High	High	High
	Insufficient Testing for Security	Moderate	Low	Moderate	Moderate
Integration/System Testing Stage	Failure to Test for Security Vulnerabilities	High	High	High	High
	Inadequate Testing of Security Features	High	High	High	High
	Security Controls Bypass	Moderate	High	High	Moderate
	Lack of Secure Data Transmission Testing	High	High	High	High
Operation/Maintenance Stage	Data Retention Risks	High	Moderate	High	High
	Unaddressed Vulnerabilities	High	High	High	High
	Inadequate Backup and Recovery	Moderate	High	High	High
	Inadequate Audit Trails	Low	Medium	Low	Low
	Weak Access Controls	Low	Moderate	Moderate	Moderate
	Insufficient Monitoring and Logging	High	High	Moderate	High

Equation (1).

$$SRA_P = f(R_i)\begin{cases} No\ Risk \\ Low\ Risk \\ Moderate\ Risk \\ High\ Risk \end{cases} \tag{1}$$

The several presumptions concerning the identified risk factors used in this investigation are listed below.

i. The threat variables that have been confirmed for each stage of the SDLC have an identical level of impact on SRA, hereafter they all have equivalent masses.
ii. The menace influences are given a value of 0 if taken into consideration (value = Yes) and a value of 1 if not (value = No); and
iii. The amount of risk variables found was used to categorize each level of safety threat (No, Low, Modest, and High). An SDLC stage with four risk factors shall consequently assign each SRA level a score of 4.

To create the two labels for each risk factor, two triangle membership functions were used. According to equations (2) and (3), the intervals for the Yes and No labels are [−0.5 0.0 0.5] and [0.5 1.0 1.5], respectively, with a clear center value of 0. (3).

$$Yes(x; -0.5, 0.0, 0.5) = \begin{cases} 0; x \le -0.5 \\ \dfrac{x + 0.5}{0.5}; -0.5 < x \le 0 \\ \dfrac{0.5 - x}{0.5}; 0 < x \le 0.5 \\ 0; x > 0.5 \end{cases} \tag{2}$$

$$No(x;0.5,1.0,1.5)=\begin{cases}0;x\le 0.5\\ \dfrac{x-0.5}{0.5};0.5<x\le 1\\ \dfrac{1.5-x}{0.5};1<x\le 1.5\\ 0;x>1.5\end{cases} \tag{3}$$

Additionally, for each SDLC stage, four triangular classifiers were used to generate four tags for each output variable of the security risk evaluation. The range for the No hazard tag is defined by Equation (4) as [−0.5, 0.5] with a sharp center value of 0. Equation (5) defines the Reduced label range as [0.5, 1.5], Equation (6) defines the Medium risk labeling range as [1.5, 2.5], and Equation (7) specifies the High-risk labeling range as [2.5, 3.5] with a sharp central value of 3.

$$No\ Risk\ (0;-0.5,0\ 0.5)=\begin{cases}0;\quad x\le -0.5\\ \dfrac{x+0.5}{0.5};\ -0.5\le x\le 0\\ \dfrac{0.5-x}{0.5};\quad 0\le x\le 0.5\\ 0;\ x\ge 0.5\end{cases} \tag{4}$$

$$Low\ Risk\ (1;0.5,1,1.5)=\begin{cases}0;\quad x\le 0.5\\ \dfrac{x-0.5}{0.5};\ 0.5\le x\le 1\\ \dfrac{1-x}{0.5};\quad 0.1\le x\le 1.5\\ 0;\ x\ge 1.5\end{cases} \tag{5}$$

$$Moderate\ Risk(2;1.5,2,2.5)=\begin{cases}0;\quad x\le 1.5\\ \dfrac{x-1.5}{0.5};\ 1.5\le x\le 2\\ \dfrac{2.5-x}{0.5};\quad 2\le x\le 2.5\\ 0;\ x\ge 2.5\end{cases} \tag{6}$$

$$High\ Risk\ (3;2.5,3,3.5)=\begin{cases}0;\quad x\le 2.5\\ \dfrac{x-2.5}{0.5};\ 2.5\le x\le 3\\ \dfrac{3.5-x}{0.5};\quad 3\le x\le 3.5\\ 0;\ x\ge 3.5\end{cases} \tag{7}$$

3.4. Simulation and validation of neuro-fuzzy SRA model

The SRA model simulation was performed using the ANFIS model development tool which is available on the MATLAB R2015a software. The toolbox requires that a dataset be provided to develop the SRA model, which was then loaded into the MATLAB SRA model editor. The dataset was split into three proportions consisting of the training, testing, and checking datasets. First, the training, testing, and checking dataset containing the historical SRA dataset were loaded into the ANFIS editor. The SRA model was trained using the training and checking dataset. Before training, the SRA model derived from the dataset using grid partition with their respective input and membership functions was defined. The training technique involved using the hybrid algorithm, which combined

Table 4
Distribution of dataset used for the simulation of ANFIS SRA model.

Proportion of Dataset	Dataset Name	Number of Rows
80 % for Training	Training1 Data	247
20 % for Testing	Testing1 Data	62
Total		309
70 % for Training	Training2 Data	216
30 % for Testing	Testing2 Data	93
Total		309
60 % for Checking	Check Data	185

least squares regression and the back-propagation algorithm following the SRA model tested using the testing dataset.

Using data collected for each SDLC phase, four simulations were performed consisting of 70 % and 80 % of datasets for training while 30 % and 20 % of each dataset respectively were used for testing each training simulation while 60 % were used for validation. Table 4 shows the distribution of the dataset collected among the different classes of the dataset created for SRA model simulation. The two sets of training and testing data pairs, namely 70/30 and 80/20 percent, alongside the checking data, at each stage of the SDLC, were produced to create their respective SRA model required for the SRA of the phase. Therefore, 25 datasets were created in all such that five were produced for each Stage of SDLC.

The SRA model simulation for each SDLC phase was done by initially importing the five datasets via MATLAB, as shown in the interface shown in Fig. 2. The datasets were initially stored in spreadsheet format as .exe files and imported to the MATLAB workspace to facilitate easy retrieval when needed by the ANFIS development GUI. The interface for the SRA model simulation was used to import the training and checking dataset required for modeling the SRA model.

As shown in Fig. 3 following data import, the asterisks (*) represent the training data while the crosses (+) represent the checking dataset. Afterward, the SRA model was formulated from the dataset using the grid partition method. The interface shown in Fig. 3 was used to formulate the SRA model using grid partition based on the dataset provided for the requirement phase of SDLC. The values of the number of membership functions required for the risk factors were presented as two alongside the type of membership function defined as *trimf* for triangular membership function while the name of the output SRA MF type was declared constant. During the validation of the ANFIS model, a confusion matrix was used to evaluate the performance of the SRA model in assessing security risks at each SDLC phase. The confusion matrix consists of a 4 by 4 matrix, which has the actual values of the number of cases for the output calculated along the rows, while the number of predictions of the cases is presented on the column as shown in Fig. 4. As shown in the diagram, the total actual cases for No risk is (A + B + C + D), for Low Risk is (E + F + G + H), for Moderate risk is (I + J + K + L) and for High risk is (M + N + O + P), while the total predicted cases for No risk is (A + E + I + M), for Low risk is (B + F + J + N), for Moderate risk is (C + G + K + O) and for High risk is (D + H + L + P). Also, A is the total number of correctly classified No risk cases, F is the total number of correctly classified Low-risk cases, K is the total number of correctly classified Moderate-risk cases, and P is the total number of correctly classified High-risk cases, while every other cell has many records incorrectly classified.

Four performance evaluation metrics were estimated from the confusion matrices that were extracted from each simulation during model validation; they are presented as follows:

i. **Accuracy** – the proportion of correct classification of the ANFIS model. The sum of the diagonals (A + F + K + P) is divided by the total cases (no, low, moderate, and high risk).
ii. **True positive (TP) rate/recall** – the proportion of actual cases predicted correctly by the ANFIS model. For example, the TP rate of no risk is the total correct classification of no risk, A divided by the total actual no cases (A + B + C + D).
iii. **False Positive (FP)/false alarm rate** – the proportion of actual cases misclassified as predicted. For example, the FP rate of No risk is the number of misclassifications (E + I + M) divided by the total cases for (Low, Moderate, and High risk).

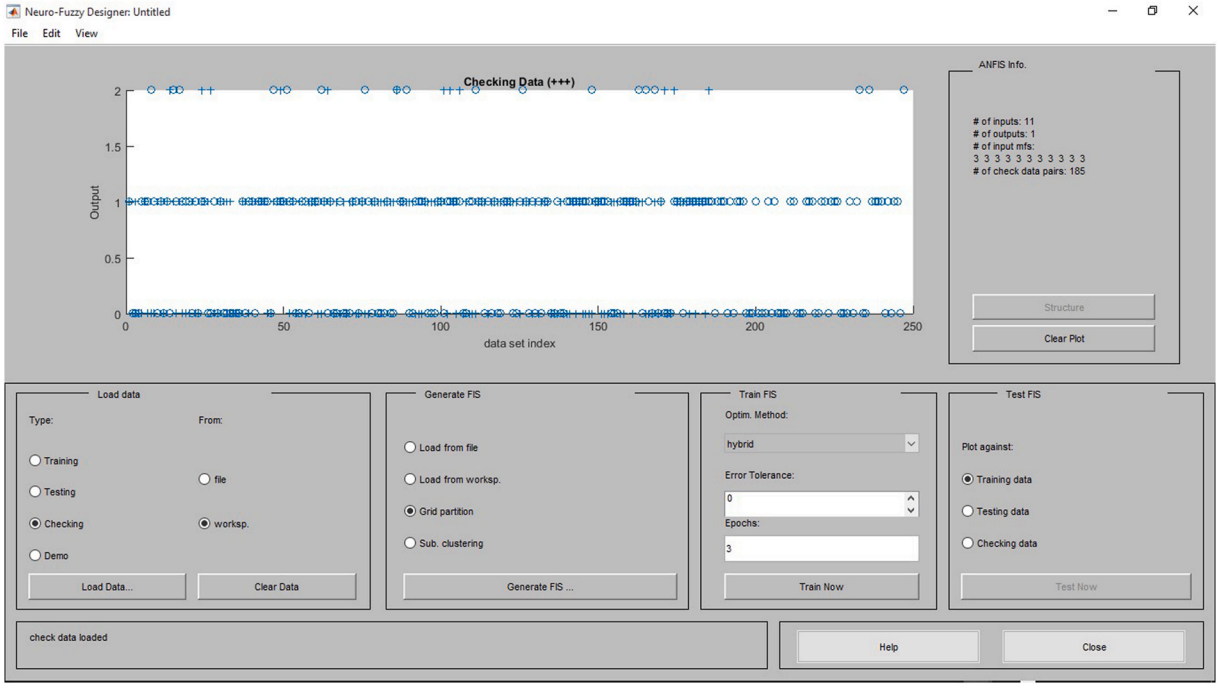


Fig. 2. Interface of data import for SRA model simulation data.

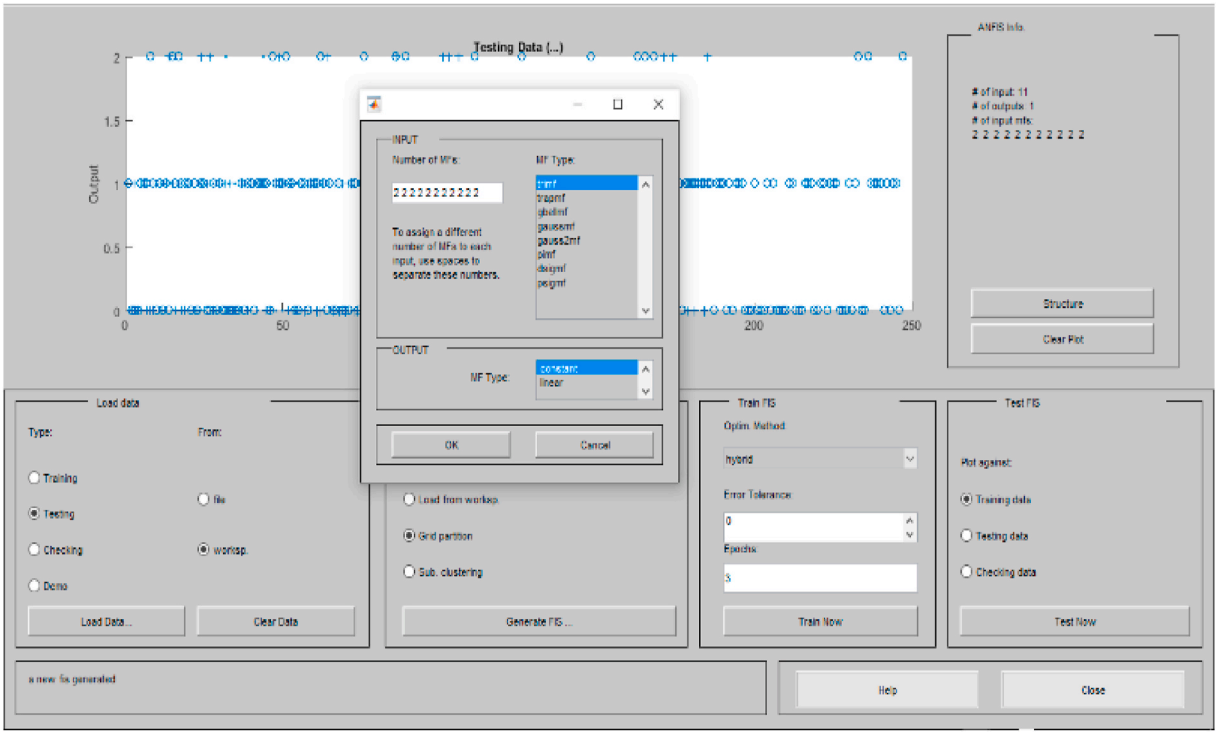


Fig. 3. Interface for a description of SRA model parameters.

W	X	Y	Z	← Predicted as
A	B	C	D	No Risk = W
E	F	G	H	Low Risk = X
I	J	K	L	Moderate Risk Y
M	N	O	P	High Risk = Z

Fig. 4. 4 by 4 confusion matrix for interpreting simulation results.

iv. **Precision** – the proportion of the predicted cases that is correct. For example, the precision for no risk is the total of correct no-risk cases (A) divided by the total predicted no-risk cases (A + E + I + M).

3.5. Design and implementation of a web-based system prototype

Following the identification of the neuro-fuzzy (ANFIS) SRA model with the best performance for each SDLC phase, a web-based implementation of a prototype system integrating the SRA models was developed using Web 2.0 technologies. A description of the system design of the prototype system using ML diagrams is presented in the following paragraphs. The system administrator was used to create the profile of authorized users by creating usernames and passwords for each authorized user. The administrator's use was performed to ensure authorized users only accessed the information stored in the system. Fig. 5 shows the use case diagram of system user action.

Following the creation of system users, the SRA assessor accesses the system by providing the administrator's username and password, thus allowing the user to access the dashboard from where the required activities were performed. Following accessing the system, the SRA assessor was led to the dashboard from where the SDLC phase is selected, the respective responses for the identified and associated risk factors are presented, following which the system uses the SRA model created by ANFIS to assess the security risk of the phase based on the response of the user. The system then returns the results of the assessment of the security risk of the respective

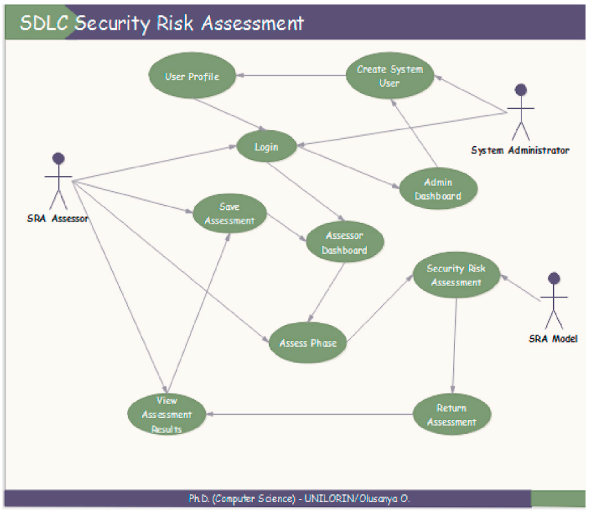


Fig. 5. Use case diagram of system user action.

phase to the SRA assessor, who also chooses to store the assessment in the database. Following the identification and specification of the user requirements of the prototype system, the data model was specified using UML class diagrams. Fig. 6 shows the class diagram that was used to specify the data model for the system. The class diagram consists of a *SYSTEM* which is composed of a *system_user* class and an *SRA_Assessment_Model* Package. The *system_user* class aggregates the *system_administrator* and the *SRA_Assessor* classes that represent data stored about the system's users. However, the *SRA_Assessment_Model* package consists of the *ANFIS_Model* and the *Result* classes required to assess the SDLC phase chosen by the SRA assessor.

Information about the SDLC phase chosen and the responses provided by the SRA assessor was handled and stored by the *SDLC_Phase* class, which presented the data to the *SRA_Assessment_Model* package. When the SRA assessor provides the data to the *SDLC_Phase* class, the data is then ported to the *SRA_Assessment_Model* package, which takes the values of the *SDLC_Phase* class and sends them to the *ANFIS_Model* class, which evaluates the security risk and returns the result to the *Result* class. The results presented can also be viewed by the SRA assessor, after which another assessment was done. The prototype web-based system was implemented using Web 2.0 technologies such as PHP, CSS, and SQL to develop an information system used to evaluate the safety risk evaluation of each SDLC phase for this study. The web-based information system took into consideration the following. Each SRA model was formulated and validated for the SRA of each stage of the SDLC process. The process was done to determine the SRA's output of each stage of the SDLC based on the inputs (Yes or No) provided by the user. The implemented prototype system provided a means via which the users of the system (community of software developers) can assess each phase of the SDLC process based on their inputs.

4. Results

After the 309 questionnaires issued among the web developers chosen for this study were collected, With a proportion of 64.7 %, it

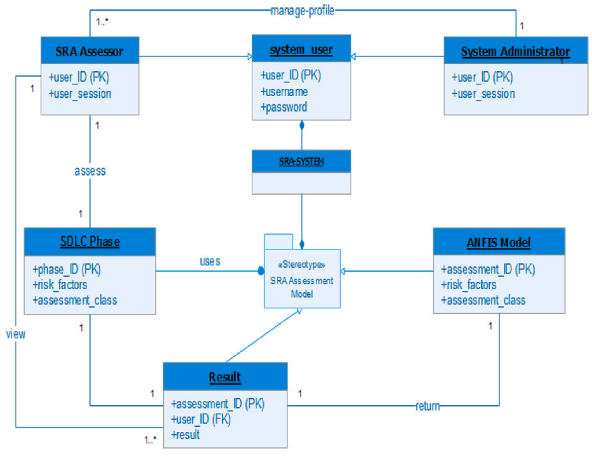


Fig. 6. Class diagram of the prototype system data model.

was noted that the majority was distributed among respondents chosen from a variety of sources who declined to be identified. Respondents chosen from PyCOM-NG and SIDMACH came in second with proportions of 9.1 % and 3.2 %, respectively, followed by APTECH with 4.9 %, TASUED with 4.5 %, CC-HUB with 3.6 %, NCS with 3.2 %, and IFE with 0.97 percent.

4.1. Results of security risk assessment of data collected

According to the data gathered most of the replies from four (4) companies were deemed to pose no potential threat during the specification stage of the SDLC, whereas the bulk of the replies from three (3) companies were judged to have minimal potential threat. Nevertheless, some of the examinations of the answers from these software development companies revealed a minor security concern. According to the data gathered for the evaluation of potential threats during the SDLC planning stage, the most of replies from most software development companies examined had no security risk, but the majority of comments from companies like SIDMACH, IFE, CC-HUB, and General had a low-security risk. However, some of the examinations of the answers from PyCOM-NG, SIDMACH, and General revealed a considerable security concern. Additionally, one respondent from each of the groups was deemed to be at high risk.

According to the data gathered the bulk of replies from TASUED, NCS, PyCOM-NG, and APTECH had no security risk throughout the SDLC implementation stage, but the majority of responses from other companies had a minimal risk. Nonetheless, some of the replies from the aforementioned companies were rated as having a medium potential threat, while a PyCOM-NG responder provided an assessment of a high-security risk. According to the data gathered, the bulk of replies from various companies examined had no potential threat throughout the SDLC implementation stage, whereas the respondents from IFE had a low degree of potential threat. Even though evaluations of CCHUB, General, PyCOM-NG, and SIDMACH replies showed a moderate potential threat, PyCOM-NG and SIDMACH respondents were found to have a high-security risk.

The findings from the analysis, the results of the vulnerabilities for the SDLC operational stage showed that a large number of reactions from the companies showed that there is no potential threat, while the majority of respondents from General had a minimal risk and an equivalent vast bulk of reactions is Low-security risks and None were present from CCHUB. Nevertheless, some of the comments from IFE, General, TASUED, SIDMACH, CCHUB, and PyCOM-NG received mixed reviews and were found to have a moderate security risk, while General and PyCOM-NG respondents were found to have a high-security risk.

4.2. The ANFIS-enabled SRA model experimental results

The SRA model was developed from the dataset using the Graphical User Interface (GUI) provided by MATLAB using the command *anfisedit* on the command-line interface (CLI). The data required for developing the SRA model was first prepared before the process of model simulation. However, the mode of data preparation was done based on a proportion of 70/30 percent and 80/20 percent of the 309 records collected for use as Training/Testing data, while the model building was complemented using checking data represented by 60 % of the dataset collected, upon starting the simulation, which was performed over 20 epochs using the hybrid of back-propagation and linear regression for model training to estimate the linear and non-linear parameters that connect the nodes of the artificial neural network with the fuzzy membership, functions created by the ANFIS model. Using 80 % and 70 % respectively for training, 60 % was used for model checking during training. The results showed the average error rates of the training data were $7.524 \times 10^{-7,}$ and $7.535 \times 10^{-7,}$ and of the checking data were 7.370×10^{-7} and 7.437×10^{-7} for the first and second simulations, respectively. Following the process of building the SRA model using the training and checking dataset, the testing dataset was also used to validate the model. The results showed that the testing error for the first and second simulations using 20 % and 30 % of the data

Table 5
Summary of model validation of ANFIS SRA model for SDLC phases.

SDLC Phase	Simulation	Correct	Incorrect	Accuracy (%)	True Positive (TP) rate	False Positive (FP) rate
Requirement Analysis/Definition Stage	Training I	247	0	100.00	1.0000	0.0000
	Testing I	47	15	75.81	0.5923	0.1613
	Training II	216	0	100.00	1.0000	0.0000
	Testing II	68	25	73.12	0.5655	0.1667
Design Stage	Training I	245	2	99.19	0.9146	0.0038
	Testing I	48	14	77.42	0.5652	0.1471
	Training II	213	3	98.11	0.8702	0.0046
	Testing II	77	16	82.80	0.4790	0.0852
Implementation/Unit Testing Stage	Training I	241	4	98.38	0.9736	0.0101
	Testing I	49	13	79.03	0.6833	0.1275
	Training II	214	2	99.07	0.9938	0.0045
	Testing II	74	9	89.16	0.9091	0.0790
Integration/System Testing Stage	Training I	243	4	98.38	0.7468	0.0051
	Testing I	61	1	98.39	0.9912	0.0074
	Training II	211	5	97.69	0.7447	0.0073
	Testing II	86	7	92.47	0.8948	0.0383
Operation/Maintenance Stage	Training I	236	11	95.55	0.7837	0.0160
	Testing I	58	4	93.55	0.6964	0.0230
	Training II	212	9	95.93	0.7372	0.0146
	Testing II	83	20	80.58	0.5816	0.0806

showed average error rates of 0.65991 and 0.70329.

4.3. Results of the ANFIS SRA model validation

Table 5 shows the validation of the ANFIS-enabled SRA mode simulations performed for all the phases of SDLC based on the dataset collected in this study. The results printed in red are the simulations with higher validation results of the testing dataset.

Table 6 shows the performance metrics of the ANFIS SRA mode simulations performed for all the phases of SDLC based on the dataset collected in this study. The results show that the ANFIS SRA model performed reasonably better with all the parameters used about 80 %, with higher validation results of the testing dataset.

The validation of the SRA model developed for the step of requirements specification and problem recognition showed that the SRA model for the first simulation using the training/testing size of 80/20 outperformed that of 70/30. Fig. 7 shows the confusion matrices that were created for the correct and incorrect classifications of the selected simulation (Simulation I) for the training (left) and testing (right) datasets. The results showed that for the training dataset; all of the 96 actual no-risk cases were correctly classified. All of the 133 actual low-risk cases were correctly classified. All of the 18 actual moderate-risk cases were correctly classified owing to an accuracy of 100 %. For the testing dataset; all of the 31 actual no-risk cases were correctly classified, 15 out of the 26 actual low-risk cases were correctly classified 11 were misclassified as no-risk cases while 1 out of the five actual moderate cases was correctly classified, and four misclassified as no risk cases owing for an accuracy of 75.8 % which outperformed the second simulation by a value of 2.59 %. It was also observed that the TP rate and precision of Simulation I was higher while the FP rate was lower than that of Simulation II. Fig. 7 shows the results of SRA model validation for the requirement phase.

The validation of the SRA model developed for the design phase in showed that the SRA model for the second simulation using the training/testing size of 70/30 outperformed that of 80/20. The average training and testing errors of Simulation 1 were higher than those of Simulation II due to an SRA model with higher accuracy. Fig. 8 shows the confusion matrices that were created for the correct and incorrect classifications of the selected simulation (Simulation II) for the training (left) and testing (right) datasets. The results for the training dataset showed that 101 out of the 103 actual no-risk cases were correctly classified, and 1 each was misclassified as low and moderate-risk cases. All of the 98 actual low-risk cases were correctly classified, and all of the 13 actual moderate-risk cases were correctly classified owing to an accuracy of 98.1 %, which was lower than that of simulation I by a value of 1.08 %. For the testing dataset; 45 out of the 47 actual no-risk cases were correctly classified while 2 were misclassified as low-risk cases; 31 out of the 38 actual low-risk cases were correctly classified while 7 were misclassified as no risk cases, and 1 actual moderate case was incorrectly classified as no risk cases. Thus, owing to an accuracy of 82.8 % it outperformed Simulation I by a value of 5.38 %. The SRA model of simulation II was selected over simulation I since the testing dataset had lower errors than simulation I.

The results of the validation of the SRA model developed for the implementation and unit testing phase showed that the SRA model for the second simulation using the training/testing size of 70/30 outperformed that of 80/20. The average training and testing errors of Simulation I was higher than that of Simulation II owing to an SRA model with higher accuracy. Fig. 9 shows the confusion matrices that were created for the correct and incorrect classifications of the selected simulation (Simulation II) for the training (left) and testing (right) datasets. The results of the training dataset showed that 79 out of the 81 actual no-risk cases were correctly classified, and 2 were misclassified as low-risk cases. Also, all of the 104 actual low-risk cases were correctly classified, and all of the 29 actual moderate-risk cases were correctly classified. Thus, owing to an accuracy of 99.1 %, which was higher than that of simulation I by a value of 0.69 %. For the testing dataset; all 45 actual no-risk cases were correctly classified, 24 out of the 33 actual low-risk cases were correctly classified while nine were misclassified as no-risk cases, and 5 out of the 15 actual risk cases were correctly classified while 10

Table 6
Summary of model performance metrics of ANFIS SRA model for SDLC phases.

SDLC Phase	Simulation	Correct	Incorrect	Precision	Recall	F-Score
Requirement Analysis/Definition Stage	Training I	247	0	1.0000	0.9923	0.9947
	Testing I	47	15	0.8913	0.9032	0.8983
	Training II	216	0	1.0000	1.0000	1.0000
	Testing II	68	25	0.8775	0.8825	0.8796
Design Stage	Training I	245	2	0.9957	0.9995	1.0000
	Testing I	48	14	0.8060	0.8321	0.8097
	Training II	213	3	0.9641	0.9721	0.9699
	Testing II	77	16	0.9007	0.9140	0.9153
Implementation/Unit Testing Stage	Training I	241	4	0.9886	0.9836	0.9900
	Testing I	49	13	0.8943	0.8967	0.8998
	Training II	214	2	0.9953	0.9985	1.0005
	Testing II	74	9	0.9444	0.9491	0.9531
Integration/System Testing Stage	Training I	243	4	0.9753	0.9748	0.9751
	Testing I	61	1	0.9815	0.9812	0.9874
	Training II	211	5	0.9667	0.9744	0.9673
	Testing II	86	7	0.9425	0.9484	0.9389
Operation/Maintenance Stage	Training I	236	11	0.9558	0.9537	0.9610
	Testing I	58	4	0.8929	0.8964	0.8930
	Training II	212	9	0.9449	0.9472	0.9446
	Testing II	83	20	0.8350	0.8316	0.8396

A	B	C	D	
96	0	0	0	A = No
0	133	0	0	B = Low
0	0	18	0	C = Moderate
0	0	0	0	D = High

A	B	C	D	
31	0	0	0	A = No
11	15	0	0	B = Low
4	0	1	0	C = Moderate
0	0	0	0	D = High

Fig. 7. Results of SRA model validation for requirement phase.

A	B	C	D	
101	1	1	0	A = No
0	98	0	0	B = Low
0	0	13	0	C = Moderate
0	0	1	1	D = High

A	B	C	D	
45	2	0	0	A = No
7	31	0	0	B = Low
6	0	1	0	C = Moderate
1	0	0	0	D = High

Fig. 8. Results of SRA model validation for design phase.

A	B	C	D	
79	2	0	0	A = No
0	104	0	0	B = Low
0	0	29	0	C = Moderate
0	0	0	2	D = High

A	B	C	D	
45	0	0	0	A = No
9	24	0	0	B = Low
10	0	5	0	C = Moderate
0	0	0	0	D = High

Fig. 9. Results of SRA model validation for implementation phase.

were incorrectly classified as no risk cases owing for an accuracy of 79.6 % which outperformed Simulation I by a value of 10.13 %. The SRA model of Simulation II was selected over Simulation I since the training and testing dataset had lower errors than Simulation I. It was also observed that the TP rate and precision of Simulation II were higher while the FP rate was lower than that of Simulation I.

The results of the validation of the SRA model developed for the integration and system testing phase showed that the SRA model for the first simulation using the training/testing size of 80/20 outperformed that of 70/30. The average training and testing errors of Simulation II were higher than that of Simulation I owing to an SRA model with higher accuracy. Fig. 10 shows the confusion matrices that were created for the correct and incorrect classifications of the selected simulation (Simulation I) for the training (left) and testing (right) datasets. The results for the training dataset showed that 155 out of the 157 actual no-risk cases were correctly classified, and 2 were misclassified as low-risk cases. Also, all of the 50 actual low-risk cases were correctly classified, all of the 38 actual moderate-risk cases were correctly classified, while the 2 actual high-risk cases were incorrectly classified as low-risk. Thus, owing to an accuracy of 98.4 % which was higher than that of Simulation II by a value of 0.69 %. For the testing dataset; all 37 out of the 38 actual no-risk cases were correctly classified while 1 was misclassified as a low-risk case, all of the 17 actual low-risk cases were correctly classified while all 38 actual moderate-risk cases were correctly classified owing to an accuracy of 98.4 % which outperformed Simulation II by a value of 5.92 %. The SRA model of Simulation I was selected over Simulation II since the training and testing dataset had lower errors than Simulation II. It was also observed that the TP rate and precision of Simulation I was higher while the FP rate was lower than that of Simulation II.

The validation of the SRA model developed for the operation and maintenance phase showed that the SRA model for the second simulation using the training/testing size of 70/30 outperformed that of 80/20. Simulation II's average training and testing errors were higher than that of simulation I owing to an SRA model with higher accuracy. Fig. 11 shows the confusion matrices that were created for the correct and incorrect classifications of the selected simulation (Simulation I) for the training (left) and testing (right) datasets. The results for the training dataset showed that 104 out of the 110 actual no-risk cases were correctly classified while 4 and 2 were misclassified as low and moderate-risk cases, respectively. Also, 93 of the 94 real low-risk cases were correctly classified. At the same time, one was misclassified as a moderate-risk case, all of the 38 actual moderate-risk cases were correctly classified, and 1 out of the 5 actual high-risk cases was correctly classified. In contrast, 1, 1, and 2 were misclassified as no, low, and moderate risk, respectively, owing to an accuracy of 95.6 %, which was lower than that of Simulation II by a value of 0.38 %. For the testing dataset; all 26 actual no-risk cases were correctly classified, 26 out of the 28 actual low-risk cases were correctly classified, while all one each was misclassified as no and moderate-risk cases owing to an accuracy of 93.6 %, which outperformed Simulation II by a value of 12.97 %. The SRA model of Simulation I was selected over Simulation II since the testing dataset had lower errors than Simulation II.

5. Discussion

The study findings revealed that there were initially roughly 93 risk factors found. The maintenance and operations stage of the

A	B	C	D	
155	2	0	0	A = No
0	50	0	0	B = Low
0	0	38	0	C = Moderate
0	2	0	0	D = High

A	B	C	D	
37	1	0	0	A = No
0	17	0	0	B = Low
0	0	7	0	C = Moderate
0	0	0	0	D = High

Fig. 10. Results of SRA model validation for integration phase.

A	B	C	D	
104	4	2	0	A = No
0	93	1	0	B = Low
0	0	38	0	C = Moderate
1	1	2	1	D = High

A	B	C	D	
26	0	0	0	A = No
1	26	1	0	B = Low
1	0	6	0	C = Moderate
0	0	1	0	D = High

Fig. 11. Results of SRA model validation for operation phase.

SDLC had 11 potential risks recognized and confirmed, while the requirements elicitation and problem definition stage had 8 potential risks recognized and affirmed, the system design stage had 9 risk factors recognized and affirmed, and the component testing process had 4 risk factors recognized and confirmed for the integration and system testing phase.

The findings demonstrated that a scoring method was developed for each stage of the SDLC using a value of 0 for yes and 1 for no and that it was utilized to select the safety threat modules to be assessed centered on the overall tally of the user's replies. A specific interval was developed for low-risk, Moderate-risk, and high-risk using the overall score while assigning a score of 0 for cases with no risk. The findings demonstrated that the SRA model was developed using two triangular association purposes with centers of 0 and 1 for Yes and No, respectively, for each risk factor, while the security risk assessment output was developed using four triangular Without No risk, Small risk, Minimal risk, and Significant risk, membership functions with centers of 0, 1, 2, and 3, accordingly.

By using the MATLAB Fuzzy Logic Toolbox, the SRA model required for assessing the security risk of each phase of the SDLC was simulated using each set of datasets extracted for each phase of the SDLC. The results showed that 2 simulations were performed such that the first involved the use of randomly selecting 80 %, 60 %, and 20 % for training, checking, and testing the SRA model, while the second involved the use of randomly selecting 70 %, 60 %, and 30 % for training, checking and testing the SRA model. The SRA model with the best performance was selected from the simulations and was adopted for assessing the security risk of the respective SDLC phase.

The results of the study further showed that following the simulation of the SRA model required for assessing the security risk of the SDLC phase, the SRA model selected for each phase was selected following comparison. The outcomes presented that the SRA model established in simulation I was selected for the prerequisite investigation and problem identification stage. The SRA model developed in Simulation II was selected for the system design phase. The SRA model developed in Simulation II was selected for the implementation and unit testing phase. The SRA model developed in simulation I was selected for the integration and system testing phase. The SRA model developed in simulation I was selected for the operation and maintenance phase.

The study also indicated that the entire rating was separated into four components, allowing for risk assessment as No, Low, Medium, or Extreme Risk for each stage of the SDLC. This was accomplished by giving values of 0 and 1 to participants' Yes and No replies to questions concerning risk variables addressed at each step. The outcomes of the study led to the conclusion that by using 2 and 4 Fuzzy triangle membership values for each intake risk factor and output SRA category, accordingly, the dataset required for the construction of the SRA models for the potential threat could be created.

5.1. Threats to both internal and external validity

Taking a closer look at construct validity, the study was not administered during software development, but after the software development stages, respondents were asked to recollect the security risk during the SDLC phases. This strategy has the potential to skew data in several ways, the most prominent of which is the fading affect bias [21,60]. In this bias, memories associated with recollections tend to fade away. The study assumes this threat, but to mitigate it, it ensured that surveys were distributed throughout a software development course. Furthermore, the questionnaire was distributed at a time when software development was not taking place. More specifically, the questionnaire was distributed to developers before and after no software development activities, with the activities being recent and impact bias being minimized in this scenario. On the other hand, the authors believe that the threat of validity stems from the fact that the activity was carried out before the developers filled out the questionnaire. Nonetheless, because the sample is made up of developers who have worked in software development organizations for at least five years, respondents have been exposed to software development activities. The study identified software developers' participants and gave them the initially identified 93 risk factors to access the questionnaires. It was unable to guarantee the data's legitimacy because the questionnaire was anonymous (as required by data collection standards). In the trials, the software developers used a collection of independent variables

that are routinely used to estimate the likelihood of threats, but their efficacy was not evaluated beforehand.

Internal validity is jeopardized because respondents may not have equal degrees of knowledge or competence. Given that responders were picked based on their knowledge and experience in every case, the authors ensured that the software developers had a similar degree of knowledge and experience. The authors anticipated two possible challenges to external validity. The first danger is the undeniably small number of software development companies; this aspect makes it difficult to generalize the study's findings. The fact that the sample was not picked at random is the second threat. Despite these risks, given the study's exploratory nature, the sampling procedure and the number of respondents may be appropriate.

The study selected only security risks for SDLC phases that are believed to be peculiar to the selected software development companies. Because all security risks for SDLC techniques support the same core aim, the authors assume leveraging additional security risks for SDLC should yield similar findings. The study addressed reusability by establishing the conditions under which the approaches can be employed and demonstrating that they can be utilized in another scenario if these assumptions are met. Our conclusion that the procedure can be used in other circumstances is questionable, as are all inductive findings. Still, we can't quantify it because we used analytic reasoning rather than statistical reasoning. As a result, the authors have demonstrated that they are concerned about the results' reliability. However, the authors state that more research is needed to ensure that the results are repeatable.

The study used their method to assess and compare confidentiality security risk twice, with comparable results in multinational software development industries where secrecy was not necessary until external authorities enforced it. The authors go on to say that the results may be transferable in the following situations: These industries developed software using the rules that guide SDLC phases and are consequently cost-conscious, so they will not strive for maximum confidentiality. This could be different in firms that value privacy, such as health care or insurance corporations, or in organizations that value confidentiality, such as the military.

6. Conclusion

The research was able to reduce the number of risk variables proposed by previous researchers to a set of risk factors that are appropriate for stakeholders in the system/software development process to assess security risk. According to the study findings, of the risk variables that were initially defined Software engineers evaluated 11, 8, 9, 4, and 6 risk factors for the SDLC phases, from the requirements stage through the operational and maintenance phases, respectively. The research established that by comparing simulations involving a percentage split of 70/30 and 80/20 percent for training/testing proportion in the datasets collected for each phase for development, a comparison of the model validation during performance evaluation was adopted as a basis for selecting the SRA model produced by the simulation with the better result. The study concluded that using the SRA model to evaluate security risk at each stage of the SDLC provided a more secure software development process. Therefore, the study recommended that SDLC stakeholders adopt the system implementation of the SRA model for assessing security risks during software development. To ensure the establishment of a successful scoring system for security risk assessment, a weighting scheme can be created by determining the impact of each risk element on the security risk posed to each phase. The study also recommends that other machine learning algorithms such as meta-heuristic nature-inspired algorithms be adopted to develop future classification models required for assessing the relationship between identified risk factors associated with security risk from available related data. There are two ways to look at the suggested model limitations. Firstly, correlations between SDLC stages were not taken into account in the proposed technique. For instance, the proportion of risk variables present at each stage of the SDLC process. Furthermore, we have not used the tool on real software projects. Even though our model is based on real empirical data and is intended to be a helpful tool for monitoring the current stage of software development, it is not intended to be used in software development. Further research will examine how learning algorithms based on historical data from prior projects can enhance the accuracy of model predictions. The application of feature selection and Bio-inspired models will be considered for future work to select the most relevant risk factors in each stage of the SDLC process.

Data availability statement

Data will be made available on request.

CRediT authorship contribution statement

Olayinka Olufunmilayo Olusanya: Validation, Methodology, Investigation, Data curation, Conceptualization. **Rasheed Gbenga Jimoh:** Supervision, Conceptualization. **Sanjay Misra:** Writing – review & editing, Writing – original draft, Validation, Methodology, Investigation, Conceptualization. **Joseph Bamidele Awotunde:** Validation, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.heliyon.2024.e33495.

References

[1] H. Hijazi, S. Alqrainy, H. Muaidi, T. Khdour, Risk factors in software development phases, Eur. Sci. J. 10 (3) (2014, January) 213–232.
[2] R.G. Jimoh, O.O. Olusanya, J.B. Awotunde, A.L. Imoize, C.C. Lee, Identification of risk factors using ANFIS-based security risk assessment model for SDLC phases, Future Internet 14 (11) (2022) 305, https://doi.org/10.3390/fi14110305.
[3] P. Ahire, J. Abraham, Secure cloud model for intellectual privacy protection of arithmetic expressions in source codes using data obfuscation techniques, Theor. Comput. Sci. 922 (2022) 131–149, https://doi.org/10.1016/j.tcs.2022.04.018.
[4] V. Maheshwari, M. Prasanna, Integrating risk assessment and threat modeling within SDLC process, 2016 international conference on inventive computation technologies (ICICT) 1 (2016, August) 1–5, https://doi.org/10.1109/INVENTIVE.2016.7823275. IEEE.
[5] J.B. Awotunde, S. Misra, A.E. Adeniyi, M.K. Abiodun, M. Kaushik, M.O. Lawrence, A feature selection-based K-NN model for fast software defect prediction, in: International Conference on Computational Science and its Applications, Springer International Publishing, Cham, 2022, July, pp. 49–61, https://doi.org/10.1007/978-3-031-10542-5_4.
[6] M.T. Baldassarre, V. Santa Barletta, D. Caivano, M. Scalera, Integrating security and privacy in software development, Software Qual. J. 28 (3) (2020) 987–1018, https://doi.org/10.1007/s11219-020-09501-6.
[7] C. Ribeiro, D. Berry, The prevalence and severity of persistent ambiguity in software requirements specifications: is a special effort needed to find them? Sci. Comput. Program. 195 (2020) 102472 https://doi.org/10.1016/j.scico.2020.102472.
[8] M. AbdulRaheem, I.D. Oladipo, A.L. Imoize, J.B. Awotunde, C.C. Lee, G.B. Balogun, J.O. Adeoti, Machine learning assisted snort and zeek in detecting DDoS attacks in software-defined networking, Int. J. Inf. Technol. 16 (3) (2024) 1627–1643, https://doi.org/10.1007/s41870-023-01469-3.
[9] R.A. Khan, S.U. Khan, H.U. Khan, M. Ilyas, Systematic literature review on security risks and its practices in secure software development, IEEE Access 10 (2022) 5456–5481, https://doi.org/10.1109/ACCESS.2022.3140181.
[10] L.N.Q. Do, J. Wright, K. Ali, Why do software developers use static analysis tools? a user-centered study of developer needs and motivations, IEEE Trans. Software Eng. 48 (3) (2020) 835–847, https://doi.org/10.1109/TSE.2020.3004525.
[11] H. Deylami, I. Ardekani, R. Muniyandi, A. Sarrafzadeh, Effects of software security on software development life cycle and related security issues, International Journal of Computational Intelligence and Information Security 6 (8) (2015) 4–12.
[12] K. Biesialska, X. Franch, V. Muntés-Mulero, Big Data analytics in Agile software development: a systematic mapping study, Inf. Software Technol. 132 (2021) 106448, https://doi.org/10.1016/j.infsof.2020.106448.
[13] M. S Siavvas, D. Tsoukalas, M. Jankovic, D. Kehagias, D. Tzovaras, Technical debt as an indicator of software security risk: a machine learning approach for software development enterprises, Enterprise Inf. Syst. 16 (5) (2022) 1824017, https://doi.org/10.1080/17517575.2020.1824017.
[14] A.E. Adeniyi, M. Olagunju, J.B. Awotunde, M.K. Abiodun, J. Awokola, M.O. Lawrence, Augmented intelligence multilingual conversational service for smart enterprise management software, in: International Conference on Computational Science and its Applications, Springer International Publishing, Cham, 2022, July, pp. 476–488, https://doi.org/10.1007/978-3-031-10548-7_35.
[15] M.F. Unuakhalu, D. Sigdel, M. Garikapati, Integrating risk management in system development cycle, International Journal of Software and Web Sciences (IJSWS) 8 (1) (2014) 1–9.
[16] K. Sahu, R.K. Srivastava, Needs and importance of reliability prediction: an industrial perspective, Information Sciences Letters 9 (1) (2020) 33–37. https://digitalcommons.aaru.edu.jo/isl/vol9/iss1/5.
[17] A. Attaallah, H. Alsuhabi, S. Shukla, R. Kumar, B.K. Gupta, R.A. Khan, Analyzing the big data security through a unified decision-making approach, Intelligent Automation & Soft Computing 32 (2) (2022), https://doi.org/10.32604/iasc.2022.022569.
[18] L. Lin, H. Bao, N. Dinh, Uncertainty quantification and software risk analysis for digital twins in the nearly autonomous management and control systems: a review, Ann. Nucl. Energy 160 (2021) 108362, https://doi.org/10.1016/j.anucene.2021.108362.
[19] P. Bannerman, Risk and risk management in software projects: a reassessment, J. Syst. Software 81 (12) (2008) 2118–2133, https://doi.org/10.1016/j.jss.2008.03.059.
[20] N. Luburić, Integration of software security design analysis to the agile development process, Универзитет у НовоМ Саду, Ph. D. Thesis, Submitted at FACULTY OF TECHNICAL SCIENCES, UNIVERSITY OF NOVI SAD, 2020 available at: https://nardus.mpn.gov.rs/bitstream/id/41849/Disertacija.pdf (last accessed: 23.06.2024).
[21] R. Colomo-Palacios, C. Casado-Lumbreras, P. Soto-Acosta, F.J. García-Peñalvo, E. Tovar, Project managers in global software development teams: a study of the effects on productivity and performance, Software Qual. J. 22 (1) (2014) 3–19, https://doi.org/10.1007/s11219-012-9191-x.
[22] C.A. Thieme, A. Mosleh, I.B. Utne, J. Hegde, Incorporating software failure in risk analysis–Part 1: software functional failure mode classification, Reliab. Eng. Syst. Saf. 197 (2020) 106803, https://doi.org/10.1016/j.ress.2020.106803.
[23] J. Masso, F.J. Pino, C. Pardo, F. García, M. Piattini, Risk management in the software life cycle: a systematic literature review, Comput. Stand. Interfac. 71 (2020) 103431, https://doi.org/10.1016/j.csi.2020.103431.
[24] D. Gupta, Software risk assessment and estimation model, Proceedings of the International Conference on Computer Science and Information Technology (2008) 963–967, https://doi.org/10.1109/ICCSIT.2008.184.
[25] T.E. Abioye, O.T. Arogundade, S. Misra, A.T. Akinwale, O.J. Adeniran, Toward ontology-based risk management framework for software projects: an empirical study, J. Software: Evolution and Process 32 (12) (2020) e2269, https://doi.org/10.1002/smr.2269.
[26] A. Gandhi, A. Naik, K. Thakkar, Gahirwal, Risk management in software development using artificial neural networks, Int. J. Comput. Appl. 93 (19) (2014) 22–27.
[27] J.B. Awotunde, F.E. Ayo, R.O. Ogundokun, O.E. Matiluko, E.A. Adeniyi, Investigating the roles of effective communication among stakeholders in collaborative software development projects, in: International Conference on Computational Science and its Applications, Springer, Cham, 2020, July, pp. 311–319, https://doi.org/10.1007/978-3-030-58817-5_24.
[28] R.N.K. Sharareh, Z. Xiao-Jun, Fuzzy logic approach to predict the outcome of tuberculosis treatment course destination, in: Proceedings of the World Congress on Engineering and Computer Science 2, San Francisco, USA, 2009.
[29] L.A. Zadeh, Fuzzy Sets, Information and Control 8 (1965) 338–353.
[30] R. Jang, C. Sun, E. Mizutani, Neuro-Fuzzy and Soft Computation, Prentice Hall, Upper Saddle River, NJ, USA, 1997.
[31] G.C. Sampada, T.I. Sake, A systematic review on security metric in secure software development lifecycle, Smart Computing (2021) 331–336, https://doi.org/10.1201/9781003167488.
[32] M.A. Ibraigheeth, S.A. Fadzli, Fuzzy logic driven expert system for the assessment of software projects risk, Int. J. Adv. Comput. Sci. Appl. 10 (2) (2019), https://doi.org/10.14569/IJACSA.2019.0100220.
[33] E. Babaie, A novel method for software reliability assessment via neuro-fuzzy system, International Journal of Reliability, Risk and Safety: Theory and Application 5 (1) (2022) 43–48, https://doi.org/10.30699/IJRRS.5.1.6.
[34] M.T.J. Ansari, D. Pandey, M. Alenezi, STORE: Security Threat-Oriented Requirements Engineering Methodology, Journal of King Saud University-Computer and Information Sciences, 2018, https://doi.org/10.1016/j.jksuci.2018.12.005.
[35] M. Siavvas, D. Kehagias, D. Tzovaras, E. Gelenbe, A hierarchical model for quantifying software security based on static analysis alerts and software metrics, Software Qual. J. 29 (2) (2021) 431–507, https://doi.org/10.1007/s11219-021-09555-0.
[36] M. GiljeJaatun, P. Håkonlvleland, Security requirements for the Rest of us, A Survey:, Software IEEE 25 (1) (2008), https://doi.org/10.1109/MS.2008.19.
[37] K. Sahu, R.K. Srivastava, Predicting software bugs of newly and large datasets through a unified neuro-fuzzy approach: reliability perspective, Adv. Math.: Scientific Journal 10 (1) (2021) 543–555, https://doi.org/10.37418/amsj.10.1.54.
[38] A. Agrawal, A.H. Seh, A. Baz, H. Alhakami, W. Alhakami, M. Baz, R.A. Khan, Software security estimation using the hybrid fuzzy ANP-TOPSIS approach: design tactics perspective, Symmetry 12 (4) (2020) 598, https://doi.org/10.3390/sym12040598.

[39] A.M. Sharif, M.Z.A. Rozan, Design and implementation of project time management risk assessment tool for SME projects using oracle application express, International Journal of Computer and Information Engineering 4 (5) (2010) 1029–1034.
[40] R.A. Khan, S.U. Khan, M.A. Akbar, M. Alzahrani, Security risks of global software development life cycle: industry practitioner's perspective, J. Software: Evolution and Process (2022) e2521, https://doi.org/10.1002/smr.2521.
[41] M. Green, M. Smith, Developers are not the enemy!: the need for useable security apis, IEEE Security & Privacy 14 (5) (2016) 40–46, https://doi.org/10.1109/MSP.2016.111.
[42] Y. Acar, S. Fahl, M.L. Mazurek, You are not your developer, either: a research agenda for useable security and privacy research beyond end users, in: 2016 IEEE Cybersecurity Development (SecDev), IEEE, 2016, November, pp. 3–8, https://doi.org/10.1109/SecDev.2016.013.
[43] H. Palombo, A.Z. Tabari, D. Lende, J. Ligatti, X. Ou, An ethnographic understanding of software (in) security and a co-creation model to improve secure software development, in: Sixteenth Symposium on Useable Privacy and Security ({SOUPS} 2020), 2020, pp. 205–220.
[44] M. Martin, B. Livshits, M.S. Lam, Finding application errors and security flaws using PQL: a program query language, ACM Sigplan Not. 40 (10) (2005) 365–383, https://doi.org/10.1145/1103845.1094840.
[45] K. Sahu, F.A. Alzahrani, R.K. Srivastava, R. Kumar, Evaluating the impact of prediction techniques: software reliability perspective, Comput. Mater. Continua (CMC) 67 (2) (2021), https://doi.org/10.32604/cmc.2021.014868.
[46] K. Sahu, F.A. Alzahrani, R.K. Srivastava, R. Kumar, Hesitant fuzzy sets based symmetrical model of decision-making for estimating the durability of web application, Symmetry 12 (11) (2020) 1770, https://doi.org/10.3390/sym12111770.
[47] M. Backes, K. Rieck, M. Skoruppa, B. Stock, F. Yamaguchi, Efficient and flexible discovery of php application vulnerabilities, in: 2017 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, 2017, April, pp. 334–349, https://doi.org/10.1109/EuroSP.2017.14.
[48] Y. Acar, C. Stransky, D. Wermke, C. Weir, M.L. Mazurek, S. Fahl, Developers need support, too: a survey of security advice for software developers, in: 2017 IEEE Cybersecurity Development (SecDev), IEEE, 2017, September, pp. 22–26, https://doi.org/10.1109/SecDev.2017.17.
[49] K. Suresh, R. Dillibabu, A novel fuzzy mechanism for risk assessment in software projects, Soft Comput. 24 (2020) 1683–1705, https://doi.org/10.1007/s00500-019-03997-2.
[50] A.M. Sharif, S. Basri, H.O. Ali, Strength and weakness of software risk assessment tools, International Journal of Software Engineering and its Applications 8 (3) (2014) 389–398. https://www.earticle.net/Article/A218558.
[51] J. Witschey, S. Xiao, E. Murphy-Hill, Technical and personal factors influencing developers' adoption of security tools, in: Proceedings of the 2014 ACM Workshop on Security Information Workers, 2014, November, pp. 23–26, https://doi.org/10.1145/2663887.2663898.
[52] N. Mustafa, N.M. Sanusi, I.E.A. Jalil, Software risk assessment visualization tool design using probabilistic inference models, International Symposium of Information Technology 1 (2010) 1–3, https://doi.org/10.1109/ITSIM.2010.5561373.
[53] I.P. Gambo, A.H. Soriyan, R.N. Ikono, A proposed process model for requirements engineering using Delphi techniques for prioritization, Int. J. Inf. Technol. Comput. Sci. 1 (2015) 73–80, https://doi.org/10.5815/ijitcs.2015.01.09.
[54] R. Pooja, S.S. Dalwinder, Neuro-fuzzy based software risk estimation tool, Global Journal of Computer Science and Technology Software and Data Engineering 13 (6) (2013).
[55] M. Batar, K.U. Birant, A.H. Işik, Development of rule-based software risk assessment and management method with fuzzy inference system, Sci. Program. 2021 (2021) 1–14, https://doi.org/10.1155/2021/5532197.
[56] P. Srisaeng, G.S. Baxter, G. Wild, An adaptive neuro-fuzzy inference system for forecasting Australia's domestic low cost carrier passenger demand, Aviation 19 (3) (2015) 150–163, https://doi.org/10.3846/16487788.2015.1104806.
[57] M.M. Ghiasi, M. Arabloo, A.H. Mohammadi, T. Barghi, Application of ANFIS soft computing technique in modeling the CO2 capture with MEA, DEA, and TEA aqueous solutions, Int. J. Greenh. Gas Control 49 (2016) 47–54, https://doi.org/10.1016/j.ijggc.2016.02.015.
[58] A. Mishra, Z. Otaiwi, DevOps and software quality: a systematic mapping, Computer Science Review 38 (2020) 100308.
[59] D. Oliveira, M. Rosenthal, N. Morin, K.C. Yeh, J. Cappos, Y. Zhuang, It's the psychology stupid: how heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots, in: Proceedings of the 30th Annual Computer Security Applications Conference, 2014, December, pp. 296–305.
[60] M. Arkhipov, E. Krueger, D. Kurtener, Evaluation of ecological conditions using bioindicators: application of fuzzy modeling, in: In International Conference on Computational Science and its Applications, Springer, Berlin, Heidelberg, 2008, June, pp. 491–500, https://doi.org/10.1007/978-3-540-69839-5_36.
[61] J. Alshudukhi, K. Yadav, Survivability development of wireless sensor networks using neuro fuzzy-clonal selection optimization, Theor. Comput. Sci. 922 (2022) 25–36, https://doi.org/10.1016/j.tcs.2022.04.008.