future internet

MDPI

Article

Identification of Risk Factors Using ANFIS-Based Security Risk Assessment Model for SDLC Phases

Rasheed Gbenga Jimoh [1], Olayinka Olufunmilayo Olusanya [2], Joseph Bamidele Awotunde [1], Agbotiname Lucky Imoize [3,4] and Cheng-Chi Lee [5,6,*]

[1] Department of Computer Science, Faculty of Information and Communication Sciences, University of Ilorin, Ilorin 240003, Nigeria
[2] Department of Computer Science, Tai Solarin University of Education, Ijagun 120101, Nigeria
[3] Department of Electrical and Electronics Engineering, Faculty of Engineering, University of Lagos, Akoka, Lagos 100213, Nigeria
[4] Department of Electrical Engineering and Information Technology, Institute of Digital Communication, Ruhr University, 44801 Bochum, Germany
[5] Research and Development Center for Physical Education, Health, and Information Technology, Department of Library and Information Science, Fu Jen Catholic University, New Taipei 24205, Taiwan
[6] Department of Computer Science and Information Engineering, Asia University, Taichung 41354, Taiwan
* Correspondence: cclee@mail.fju.edu.tw

Abstract: In the field of software development, the efficient prioritizing of software risks was essential and play significant roles. However, finding a viable solution to this issue is a difficult challenge. The software developers have to adhere strictly to risk management practice because each phase of SDLC is faced with its individual type of risk rather than considering it as a general risk. Therefore, this study proposes an adaptive neuro-fuzzy inference system (ANFIS) for selection of appropriate risk factors in each stages of software development process. Existing studies viewed the SDLC's Security risk assessment (SRA) as a single integrated process that did not offer a thorough SRA at each stage of the SDLC process, which resulted in unsecure software development. Hence, this study identify and validate the risk factors needed for assessing security risk at each phase of SDLC. For each phase, an SRA model based on an ANFIS was suggested, using the identified risk factors as inputs. For the logical representation of the fuzzification as an input and output variables of the SRA risk factors for the ANFIS-based model employing the triangular membership functions. The proposed model utilized two triangular membership functions to represent each risk factor's label, while four membership functions were used to represent the labels of the target SRA value. Software developers chose the SRA risk factors that were pertinent in their situation from the proposed taxonomy for each level of the SDLC process as revealed by the results. As revealed from the study's findings, knowledge of the identified risk factors may be valuable for evaluating the security risk throughout the SDLC process.

Keywords: fuzzy logic; software product; software development life cycle; inference systems; security risk assessment; adaptive neuro-fuzzy

Citation: Jimoh, R.G.; Olusanya, O.O.; Awotunde, J.B.; Imoize, A.L.; Lee, C.-C. Identification of Risk Factors Using ANFIS-Based Security Risk Assessment Model for SDLC Phases. *Future Internet* **2022**, *14*, 305. https://doi.org/10.3390/fi14110305

Academic Editor: Wei Yu

Received: 19 September 2022
Accepted: 20 October 2022
Published: 26 October 2022

1. Introduction

Software development is one of the biggest industries globally. Several software initiatives of different scope, expense, and complexity are being developed [1–3]. The Software Development Life Cycle (SDLC) is a framework or method that a software organization imposes on the process of producing a software product [4,5]. There are various stages in SDLC for the development of software products such as issue attribution, viability of the project investigation, requirement specification, design, development, testing, deployment, and maintenance are the standardized life cycle for software development [6–9]. The three main goals of an SDLC are to create high-quality software products, offer appropriate administration throughout the production of software product, and increase the effectiveness

of the software organization workforce [10,11]. How effective are the software security risks can be managed throughout the SDLC process. How these risks can be linked to each stages of SDLC for the production of a secure, error-free, and effective software product within the predicted project evaluation and cost [12,13].

The steps involved in risk management are risk identification, analysis, mitigating risk, and evaluation. However, a secure software development process depends on the SDLC's inclusion of risk management [14,15]. Risk management in SDLC process become problematic because software development is not a straightforward process, according to authors in [16]. Security risk, which is considered to be the net negative effect of exercising susceptibility likelihood and incidence consequence. This is still in a developing stage in the area of software development process. Organizations can balance the needs for securing and producing a good software product using risk management process throughout software design. This will reduce the resources and the cost associated with the security measures and prevention applied throughout the SDLC [17,18]. As more people utilize and adopt software applications, and devices powered by software products, security become a crucial factor to take into account in the SDLC [19,20]. Additionally, being aware of related security concerns will make it easier to spot a defect software product [21,22].

By recognizing security needs early in the development of software product and implementing them all through the SDLC processes will surely help manufacturing company to produce a secure and cost-effective software product for their clients. This will help to always put security considerations that will help in accomplishing a secure software product for delivery within the software manufacturing company. No matter which or where SDLC phase is being considered, the risk management technique is the same, according to authors in [10]. The study explained that the SDLC is not intended to be disturbed or given additional phases, but to add security practices to an already-in-use SDLC approach. Developers must closely follow risk management procedures throughout the whole SDLC cycle in order to deliver secure software product [23,24]. Software life cycle that has security risk in mind will easily identified source code that contains some of the vulnerabilities. However, this represents a relatively little fraction of the overall risk that are necessary to take into account for both development process and human-related risks. Each of these steps comes with a unique set of risks and dangers for both software product and human being. Consequently, it is crucial to pay attention to the degree of security risk defenselessness, rather than after the software product has been released and deployed [25].

To help software developers, numerous tools, models, and techniques have been developed in the past. These tools and models have been used by project manager within various organizations during the SDLC processes to perform risk management tasks [26,27]. However, the majority of the tools models used have their challenges and flaws, ranging from the inability to update the risk factors, to the failure to recognize the risk factors connected to software product issues [28,29]. Hence, an intelligent system is required that will combine risk management processes into all stages of the SDLC in the production of software products [30]. Each phase of the SDLC has a unique set of risks attached to it [31–33], but there no or little tools or models that can help developers to carry out risk management tasks across the SDLC process' phases [34,35]. Therefore, this study proposes an adaptive neuro-fuzzy system (ANFIS) that will rely on risk factors chosen from a catalogue of software risk factors checklists encountered by software developers in each stage of the SDLC software development so has to effectively consider these security risks.

The followings are the major contributions of the proposed model:

- the risk factors in SDLC phases were identified and validated to establishes the security in software development.
- the identified risk variables were used as inputs for each SDLC phase in an ANFIS based SRA model.
- the logical representation of the fuzzification of the input and output variables of the ANFIS SRA model was done using triangular membership functions.

- the proposed system was evaluated using various performance metrics and compares with existing methods.

2. Literature Review

Security is becoming a crucial factor to take into account in the SDLC due to the rising use and adoption of software applications, ICT, and devices that are powered by software applications. Additionally, knowledge of related security issues will help in spotting secured applications. Software development risks have a significant impact on the success of the final product. Developers must closely adhere to risk management practices throughout the SDLC in order to provide secure software. Therefore, an intelligent and efficient web-based expert system required for managing software risk will assist software developers and project managers in carrying out risk management-related activities and in making the decisions affecting SDLC. There are various technique used for security risk analysis, a few of them are discuss in this subsection:

2.1. Types of Software Test Techniques to Improve Trustworthiness of Security Assessment

Intrusion Modes and Effects Criticality Analysis (IMECA): The US military made a commitment in the late 1940s to switch from a strategy of "detect failure and fix it" to one of "predict failure and prevent it [36]." The techniques created centered on identifying qualitative and quantitative risks in order to avoid failure. A technique called Failure Mode, Effects & Criticality Analysis (FMECA) includes quantitative failure analysis. The FMECA entails establishing a number of connections between anticipated failures (Failure Modes), their effects on the mission, and their causes (Causes and Mechanisms). The FMECA's methodologies and practices were made public in a number of Military Standards. The most well-known of these standards, MIL-STD-1629A, is still in use today [37].

An IMECA is a variation to the FMECA-technique that considers potential system invasions. The technique can be used to different kinds of vulnerabilities, but its primary use is to evaluate technical vulnerabilities [38]. At each level of the SDLC, where risk factors and security are evaluated, for safety-related sectors, IMECA can be utilized in addition to standardized FMECA, because if such systems are breached, any vulnerability could result in a failure. It is necessary to develop a security criticality matrix in order to analyze the cyber security concerns [39]. Thus, depending on the breadth of the assessment, the following situations are conceivable:

1. A selection of specific IMECA tables are used to evaluate the complicated product or system as a whole (which are a collection of discrepancies that represent all the identified gaps) should be added to the IMECA table's one global representation of the entire system. In this instance, a global criticality matrix is built using each row of the global IMECA table as a foundation [40].
2. Evaluation of specific (sub-)systems inside the complex system/product: An adequate set of local criticality matrices can be made to correspond to certain (sub-)systems, based on a number of neighborhood IMECA tables [40].

Attack Tree Analysis: Attack tree analysis was introduced in 1999 as a method for listing different attack pathways to accomplish an attack objective [41]. The assault goal is stated at the top of the tree, and branches are a list of the sub-goals that are listed below. Each branch has a number of leaf nodes, or actions, that must be taken in order to accomplish the sub-goals and, eventually, the main objective [41]. An arc indicating "AND" logic connects leaf nodes that must be executed simultaneously to accomplish a sub-goal. In contrast, leaf nodes that indicate different ways to achieve a sub-goal are not coupled, illustrating "OR" logic. Attack trees are derived from fault tree analysis, a technique used in the aerospace industry to identify defects in intricate systems. In order to identify the failure modes and dangers for intercontinental ballistic missiles (ICBMs), Bell Telephone Laboratories created fault trees in 1962 [41,42]. Attack trees have come under fire for being static and arbitrary, despite being very helpful for a quick risk assessment [42]. Researchers have standardized attack tree components to lessen their subjectivity, which makes it easier

to compare attack trees across many systems, scale them over numerous systems, and automate attack trees using AI planning logic [41]. However, there are other structural variants of attack trees that would be relevant and appropriate for security assessments.

Vulnerability/Fault Injection Testing: This is a software testing approach that involves inserting errors into the code to increase coverage. It is sometimes combined with stress testing to determine how robust the generated software is [43]. To test and strengthen a system's stability and dependability, mistakes and defects are purposefully introduced into it. Over time, it is intended to strengthen the system's design for resilience and performance under intermittent failure scenarios. With the goal of "embracing failure" as a part of the development lifecycle, fault injection methods are a way to improve coverage and verify software resilience and error handling, either at build-time or at run-time. These techniques help engineering teams account for known and unknowable failure conditions, architect for redundancy, use retry and back-off mechanisms, etc. when developing and continually validating for failure [44]. As a result of fault injection's effectiveness, the Microsoft Security Development Lifecycle mandates fuzzing at each untrusted interface of every product as well as penetration testing [45]. This involves making errors in the system in order to find potential vulnerabilities brought on by coding mistakes, system configuration issues, or other operational deployment issues. Automated fault injection coverage in a CI pipeline encourages a Shift-Left strategy of testing for probable problems earlier in the lifecycle [46].

2.2. Related Work on Security Risk Assessment Tools for Software Test Techniques

The authors in [5] used various security risk discovered as risk factors to evaluating software development risk across every step of the SDLC procedure. The foremost aim of the study was to identify the most security risks with relation to assessing security risk at each level of the SDLC process, out of the 93 risk categories. The study did not propose how security risk assessment can be performed based on data regarding the recognized risk factors. In [12] the authors applied Artificial Neural Networks (ANN) to develop risk management in SDLC process in the development of software product. The study identified ten most relevant risk factors that can be encountered during various stages of SDLC process in evaluating security risk by software developers. The study collected relevant data associated with assessing the security risk associated with software development based on the 10 risk factors. The study was limited to the identification of 10 risk factors which could only be used to evaluate security risk when the SDLC process is complete. The authors in [34] developed a fuzzy expert system that focused on determining the overall risk of a software project. The study did not take into consideration the identification of any risk factor. The study made no provision for facilities required for adding, removing, update or prioritizing risk, and the model was simulated using MATLAB software.

Software risk management is a proactive decision-making process that includes techniques, strategies, and tools for controlling risks in software development projects. Many existing strategies for software project risk management are non-reusable and non-shareable with only textual documentation with varying perceptions. In [35], the authors proposed a life-cycle approach to ontology-based risk management framework for software projects. It is based on a dataset culled from the literature, domain experts, and practitioners. A total of 19 software specialists refine the identified risks, which are then theorized, modeled, and built using Protégé. A version method as well as a qualitative analysis and prioritization of the risks was adopted using real-life software projects. The extraction tool's performance is validated using precision recall and F-measure metrics, while the extraction tool's performance is validated using the performance metrics. The performance appraisal form and the technology acceptance model are used to assess performance and perceptions, respectively. The assessment concept scale is used to compare mean scores from performance and perception evaluations. The results revealed that costs were decreased, high-quality projects were delivered on schedule, and software engineers regarded this framework to be a useful tool for their day-to-day software development activities.

Similarly, the authors in [47] conducted a systematic literature review (SLR) on software risk in order to characterize and present the state of the art in this field, as well as to identify gaps and possibilities for future study. The scientific community's interest has shifted away from the definition of research activity that addressed an integrated risk management process in favor of work that focuses on distinct activities of this process, according to the analysis of the results of this SLR. It was also obvious that there was a distinct lack of scientific rigor work in the area of validation process in their investigations as well as a lack of usage of standards or de facto models to describe them.

In [48], the authors conducted an SLR on risk management's software process simulation modeling (SPSM), the purpose of which was to: (i) determine where this issue has been used the most in risk management efforts, (ii) view the most common forms of risk for which SPSM has been used, (iii) determine which SPSM models are most commonly utilized in risk management, (iv) determine which SPSM standards are most commonly utilized in risk management, and (v) examine how SPSM approaches and models in risk management that have been implemented in practice.

The authors in [19] developed a software risk estimation tool. The study identified 17 risk factors that are related to specialists' assessments of the security risk. The study collected associated data, after that, a security risk assessment model known as the ANFIS was developed based on the 17 risk categories identified throughout the SDLC process. The study was only focused on the creation of a security risk assessment tool using a neuro-fuzzy (ANFIS) model; however, it was limited to the use of 17 risk factors adopted at the end of the SDLC process. The authors in [49] worked on developing a qualitative risk assessment tool for offshore projects. The study adopted the use of risk factors that could only be identified systematically from the project document. The study didn't take into account ranking risks in order of importance or identifying risks specific to software development. In [50], the authors developed a fuzzy-based expert system for the estimation of risk of project failure. The study estimated various levels of risk categories and severity of impacts as a basis of estimating project failure. The study did not consider the identification of any risk factor in the estimation of project failure.

Internet-based, web-data servers, online services, and GUI-based applications are increasingly being exploited as software threats. Each phase prior to the completion of the software product requires a different form of threat modeling [51]. The most important role in the SDLC is identifying software and hardware threats [52,53]. To reduce risks, the threat modeling approach is incorporated into the SDLC early enough [54,55]. In [54], the authors proposed a Secure (S-SDLC) through Agile Methods. The study shows that vulnerabilities can be quickly identify if security is taken into consideration from the beginning with no additional time or costs to the client, and can be fixed within the project's allotted period. Additionally, it increases the likelihood of success in terms of both consistency and effectiveness.

Finding the risk factors is crucial in the initial phases of the SDLC process. Threat classification of risk-based techniques is investigated by authors in [56] to review threat attacks with regard to security issues. During the life cycle stages' complexity is used to categorize the most risk factor. The results of the suggested method are compared to those of Microsoft Stride in order to identify component boundaries, rank attacks, and better comprehend software development and operation threats in the software development process. The authors of [57] proposed a novel technique for selecting the optimal reliability prediction model. This approach combines the analytical hierarchy method (AHP), hesitant fuzzy (HF) sets, and a methodology for ranking preferences according to how closely they resemble the ideal answer. Procedural sensitivity testing was also done to validate the results using the various rounds of the method. The developers will be able to estimate reliability prediction depending on the software type with the aid of the results of the software reliability prediction models prioritizing.

From the reviewed literature, the SRA challenges attracted the attention of researchers, and ML-based techniques were used in this field to satisfy this requirement. More notably,

in the area of SRA in SDLC process in the development of software product effort. In the literature various approaches have been used like ANNs, optimization, and fuzzy methods. Therefore, considering this situation, the study proposed an integrated framework for evaluating software project risks that is based on adaptive neuro-fuzzy inference system. This study extends the work done in the literature by figuring out the dangers that each stage of the SDLC process in developing countries. This study also proposes an ANFIS model which adopts information about the identified risk factors for security risk assessment at their respective SDLC phase.

3. Materials and Methods

This study focused on categorizing the various risk factors at each stage of software development, security risk is evaluated based on information provided by software developers selected in Nigeria software development organization. These risk factors were selected from an initial number of risk factors listed in the software risk taxonomy in the literature. This study proposed an ANFIS model for evaluating software project risks in each stage of software development stages based on data gathered on the risk variables involved. A conceptual description of the various activities completed in this study is shown in Figure 1.

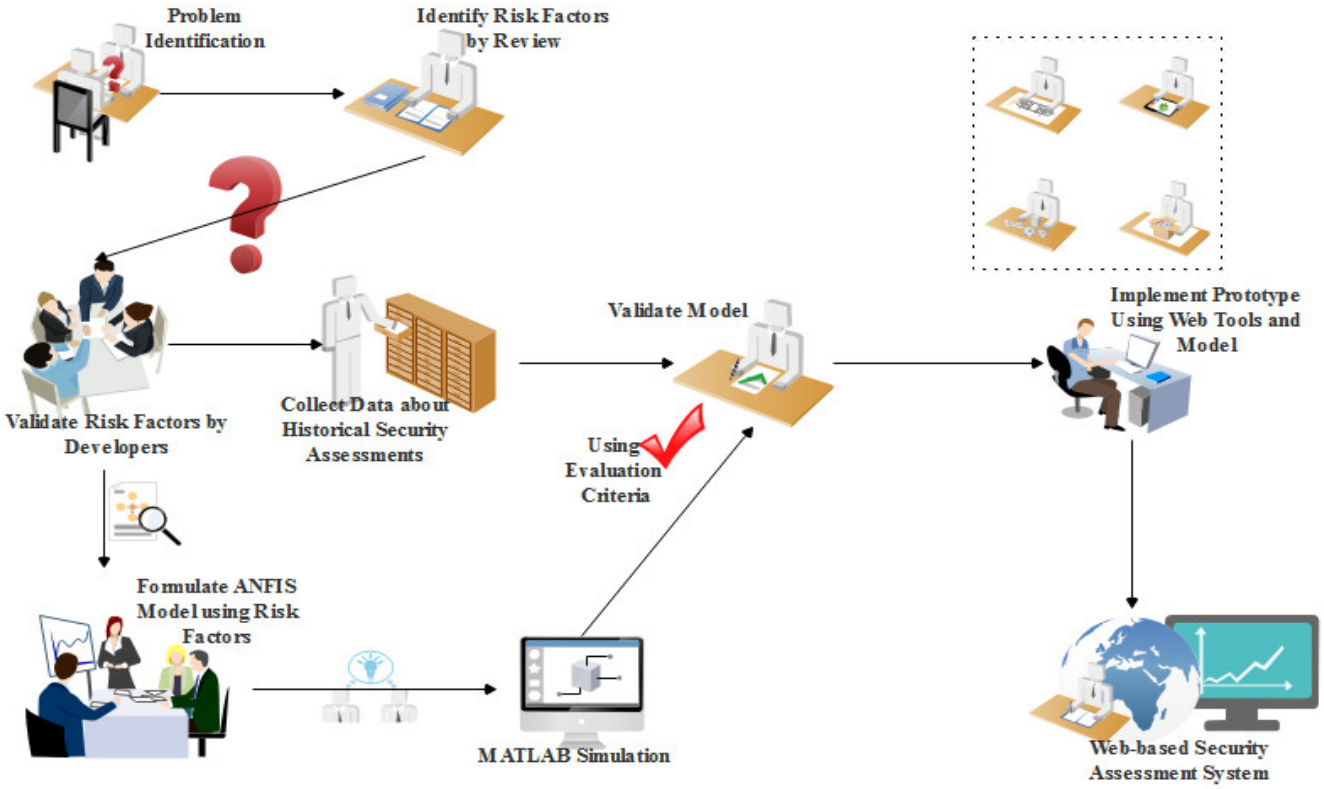

Figure 1. Conceptual view of research methodology process.

The study's first step was to pinpoint the issue of security risk factor at each stage of software development process, which was discovered through a search of relevant literature. The prominent risk indicators connected to each stage of the SDLC procedure that were discovered were validated by a number of software developers selected in Nigeria. Relevant information about the adoption of each risk factor was collected from the software developers. Finally, the adaptive neuro-fuzzy SRA model was formulated for each phase of the SDLC alongside the number of the inference rules of each SRA model stated. In most security assessment systems developed in the literature, the majority of efforts have been concentrated on evaluating security risk after system implementation. Finding the stage that is related to or directly accountable for increasing the estimated security risk is difficult.

Identifying problems, doing feasibility studies, analyzing specifications, designing the system, putting it into practice, testing it, deploying it, and maintaining it are all steps in the consistent life cycle of system development. The effectiveness of any proposed system depends greatly on each of these phases. Therefore, rather than waiting until the system is fully deployed, risk factors should take into account how much it exposed to at each stage.

3.1. Security Issues and Measures for Each SDLC Phases

Even with the usage of external perimeters like firewalls, proxies, intrusion detection systems, and antivirus software, most software cannot withstand security threats. The cause is that software inherently lacks security and has flaws that can be used in security assaults. These vulnerabilities are primarily brought about by specification, design, and coding errors that software engineers unintentionally introduce [4]. Security was regarded as a non-functional need in the SDLC because to the utilization of outsider perimeters up until this point [6]. Secure software, however, is software that is inherently resistant to attack [4]. These vulnerabilities which are mostly code and design flaws, go unreported by software experts. Software security must be built into the software in order to create one that is secure. Software can be made secure by maintaining the assets' non-repudiation, availability, confidentiality, and integrity, and assets, resources, and even the running programs themselves that the software generates, stores, processes, or sends.

In the software development life cycle, security plays a very important role, and software security testing is an important means to achieve goal of SSDLC.

The first phase of the SDLC involves identifying the need, purpose and scope of the software following which requirements proposed. At this phase, the identified risks that are used in developing the software requirements including the security requirements so that is ensured that threats and potential functionality and integration constraints considered in line with the requirements. The second phase of the SDLC involves designing the system based on the requirements. Identified risk can be used to support security analyses of the software. These risks may have impact on the architecture and design of the software. Security requirements were analysed, the security architecture were designed while functional and security tests completed.

The third phase of the SDLC involves configuring, testing and verifying modules or software components for production. Risk management supports the assessment of the software by comparing software implementation against requirements and within the operational environment. Management of identified risks were decided before software was moved for operation. The software was integrated into the software environment, security controls were tested, and the accreditation was completed.

The fourth phase of the SDLC involves deploying the software in the environment for modification via potential hardware and other (code) changes or additions as well as by changes to organisational practices. At this phase, risk management activities were performed in line with software being re-authorised and monitored for performance in a periodic process. The software is re-assessed when the IT has faced major changes in the operational environment such as new features were developed and tested, or new hardware is added or replaced. The software's operational readiness was reviewed, the system configuration was managed, and the process and procedures for monitoring are set up. The fifth phase of the SDLC involves moving the system for deployment in a real environment. At this phase, risk management activities are performed for disposable or replaceable software components so that disposal was performed properly, and that residual data was handled appropriately and migration to new system happens securely.

3.2. Identification of Risk Factors of SDLC Phases

According to authors in [5], several risk factors were identified in the software risk factor taxonomy for each of the phases in the process. In order to narrow down the essential yet crucial risk factors, questionnaires were created for Nigerian software engineers to validate the risk variables pertinent to the SDLC from the 93 risk variables that were

originally identified. Every risk factor was assessed utilizing a relevance metric with the options of Yes or No (if regarded) or No (if not regarded). The Google Forms® program was used to create the questionnaires, an extra benefit offered by Google® for users of GMail® for conducting the poll. The programmers were chosen among Nigerian programmers, picking from various phases of software development processes.

The risk variables that were taken into account throughout the level of the SDLC for requirement analysis and definition are described in Table 1. There were 21 risk factors in all, and these were groups into 5 distinct activities each with related risk factors. The system design phase is described in Table 2. A total of 19 risk factors were divided into seven distinct activity categories, each of which included a number of related risk factors. The risk variables that were taken into account during the SDLC's implementation and unit testing phases are described in Table 3. There are 23 risk factors in all, which were divided into two main groups of activities made up of many related risk factors. The risk factors that were taken into account during the SDLC's integration and system testing phase are described in Table 4. There are a total of 16 risk variables, which were divided into 5 separate activity groups made up of a number of related risk factors. Table 5 shown the risk factor in the operation and maintenance during software development processes. There are a total of 14 risk variables, which were divided into four separate activity groups.

Table 1. The requirement definition and analysis stage risk factors.

Risk Factor Classification by Activity	Number of Instances
Feasibility Study	5
Requirements Elicitation	7
Requirements Analysis	5
Requirements Validation	2
Requirements Documentation	2
TOTAL	21

Table 2. The system design stage risk factors.

Risk Factor Classification by Activity	Number of Instances
Examining Requirement Documentation	1
Choosing Architectural Design Method	1
Choosing Programming Language	1
Constructing Physical Model	5
Verifying Design	3
Specifying Design	4
Documenting Design	4
TOTAL	19

Table 3. The implementation and unit testing stage risk factors.

Risk Factor Classification by Activity	Number of Instances
Coding	13
Unit Testing	10
TOTAL	23

Table 4. The risk factors in integration and system testing stage.

Risk Factor Classification by Activity	Number of Instances
Integration	3
Integration Testing	5
System Testing	8
TOTAL	16

Table 5. Risk factors in Operation and maintenance phase.

Risk Factor Classification by Activity	Number of Instances
Installation	3
Operation	2
Acceptance Testing	6
Maintenance	3
TOTAL	14

Following the software developers' assessment of the pertinent risk factors connected to SRA at each SDLC phase in the Nigerian setting, a follow-up questionnaire was created to gather data regarding the risk variables that were taken into consideration (or not considered) throughout the various SDLC process phases. This data was utilized to create the historical dataset required to train the ANFIS model that was proposed in this work.

3.3. Validation of Risk Factors from Software Developers

The electronic questionnaire using Google Forms® was used to collect information required to validate risk factors relevant to SDLC from selected software developers. As seen on the Google Forms®, the software engineers were given the electronic questionnaire by having it sent to their mail, or by including a link in their message that directs respondents to the survey using the link displayed.

3.4. Formulation of ANFIS SRA Model

To formulate the SRA model, the validated risk factors for each SDLC phase were used as inputs for each SDLC phase. Hence, five ANFIS SRA models were formulated, one for every SDLC process phase. Each step's specific risk criteria related to the degree of security risk are accepted as inputs by the SRA model that was developed for that stage. While the vulnerability risk evaluation level at the point the SRA model was developed served as the output. Supposing P is a stage in software development life cycle, R_i is the various factor of risks connected to the SDLC phase's security risk assessment. As a result, the risk variables related to each phase SP affect the SRA at each of those phases. Consequently, equation can be used to represent the SRA at each step, as shown in Equation (1). The equation demonstrates that SRA can be used to assess any of the four recognized risk levels as shown in Equation (1).

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

Following are the various assumptions that were made about the identified risk factors adopted in this study. The risk indicators that have been verified during each SDLC phase have the same degree of effect to SRA; hence they all carry equal weights. The risk factors are assigned the value of 0 if considered (value is Yes) and one if not considered (value is No). The classification for each level of security risk assessed (No, Low, Moderate and High) was made from the number of risk factors identified. Therefore, an SDLC phase with four risk factors will divide a score of 4 across the SRA levels. As a result of this assumption, two triangular membership functions were used to formulate the two labels of each risk factor. As proposed by Jimoh and Olusanya (2019), the interval defined for the Yes label is [−0.5, 0.0, 0.5] with a crisp central value of 0, and for the No label is [0.5, 1.0, 1.5] with a central crisp value of 1 as shown in Equations (2) and (3).

$$Yes(x; -0.5,\ 0.0,\ 0.5) = \begin{cases} 0; x \leq -0.5 \\ \frac{x+0.5}{0.5}; -0.5 < x \leq 0 \\ \frac{0.5-x}{0.5}; 0 < x \leq 0.5 \\ 0; x > 0.5 \end{cases} \quad (2)$$

$$No(x; 0.5,\ 1.0,\ 1.5) = \begin{cases} 0; x \leq 0.5 \\ \frac{x-0.5}{0.5}; 0.5 < x \leq 1 \\ \frac{1.5-x}{0.5}; 1 < x \leq 1.5 \\ 0; x > 1.5 \end{cases} \quad (3)$$

Also, the four labels of the output variable for each SDLC phase of the security risk assessment were created using four triangle membership functions. The interval defined for the No risk label is [−0.5, 0.5] with a central crisp value of 0 (Equation (4)), for the Low-risk label is [0.5, 1.5] with a central crisp value of 1 (Equation (5)), for the Moderate risk label is [1.5, 2.5] with a central crisp value of 2 (Equation (6)) and for the High-risk label is [2.5, 3.5] with a central crisp value of 3 (Equation (7)). The four triangle membership functions that were utilized to create the formula are depicted in a schematic in Figure 2. The names of each ANFIS-based SRA model that has been proposed for each SDLC phase's output target variables.

$$No\ Risk\ (0; -0.5,\ 0,\ 0.5) = \begin{cases} 0;\ x \leq -0.5 \\ \frac{x+0.5}{0.5}; -0.5 \leq x \leq 0 \\ \frac{0.5-x}{0.5};\ 0 \leq x \leq 0.5 \\ 0;\ x \geq 0.5 \end{cases} \quad (4)$$

$$Low\ Risk\ (1; 0.5,\ 1,\ 1.5) = \begin{cases} 0;\ x \leq 0.5 \\ \frac{x-0.5}{0.5};\ 0.5 \leq x \leq 1 \\ \frac{1-x}{0.5};\ 0.1 \leq x \leq 1.5 \\ 0;\ x \geq 1.5 \end{cases} \quad (5)$$

$$Moderate\ Risk(2; 1.5,\ 2,\ 2.5) = \begin{cases} 0;\ x \leq 1.5 \\ \frac{x-1.5}{0.5};\ 1.5 \leq x \leq 2 \\ \frac{2.5-x}{0.5};\ 2 \leq x \leq 2.5 \\ 0;\ x \geq 2.5 \end{cases} \quad (6)$$

$$High\ Risk\ (3; 2.5,\ 3,\ 3.5) = \begin{cases} 0;\ x \leq 2.5 \\ \frac{x-2.5}{0.5};\ 2.5 \leq x \leq 3 \\ \frac{3.5-x}{0.5};\ 3 \leq x \leq 3.5 \\ 0;\ x \geq 3.5 \end{cases} \quad (7)$$

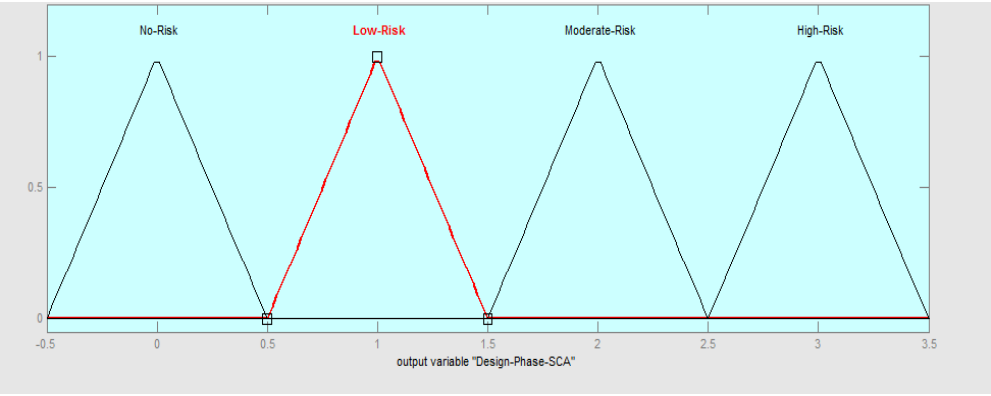

Figure 2. The triangular membership function for SRA labels.

4. The Experimental Results

This section presents the validated various risks relating to the process of evaluating security risk based on information provided by the respondents in software development industries. The respondents selected the risks factors from the 93 used as the baseline risk categories using the questionnaires based on the background of software products designs in Nigeria contents as case study on under developing countries. We identified the risk indicators that were universally accepted by all developers using frequency distribution

tables. Non-parametric tests were used to examine substantial discrepancies in the developers' answers for those that the developers did not generally agree upon. Information about the follow-up questionnaire used to collect historical data required for the development of the ANFIS-based SRA model was also demonstrated for evaluating security risk at every stage of the SDLC.

4.1. The Results Regarding Software Development's SRA Risk Factor Validation for Each Stage

Upon the distribution of the designed questionnaire to the respondents, the respondents answered to the survey by providing details on the risk factors for software development process, and provided the most suitable factors for evaluating the risks in Nigeria context. Figure 3 displays a screenshot of the survey replies provided by Google Sheets®.

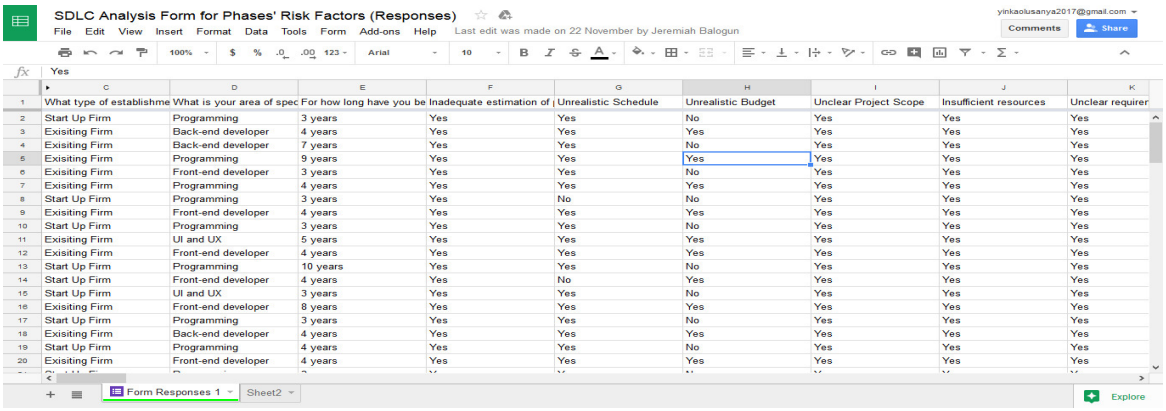

Figure 3. Responses to administered questionnaires via Google Forms.

The risk factors that respondents did not select were not considered. Nevertheless, the findings of the percentage of developers that concur with the choice of risk factors for each stage was utilized to divide the risk factors into these three (3) groups:

i. The Risk Factors in High Priority: The risk elements that were deemed crucial for assessing the security risk of the SDLC process by all developers (100%); considering that there was no variation in the developers' selection of the risk factors, these risk factors were taken into account but were not statistically examined;
ii. The Risk Factors in Low Priority: The risk categories that were not unanimously deemed necessary for each developer to evaluate the security risk of the SDLC process (<100%), non-parametric tests were used to determine the variations in responses of the developers depending on the types of organizations and specialty after these variables were eliminated but still taken into consideration for statistical analysis; and
iii. The Risk Factors in No Priority: The risk indicators that the developers chosen for this study did not concur with were eliminated and not taken into account for statistical analysis.

Table 6 displays the findings for the high priority risk categories that each software developer chosen using the designed questionnaire. From the initial 21 risk factors discovered, the findings of the selection of the pertinent risk factors taken into account throughout the requirements and definition stage revealed that 11 were deemed to be of high priority, 8 were deemed to be of low priority, 2 were regarded to be no-priority risk factors. The findings also indicated that, with the exception of unrealistic schedules, there was no statistical difference in replies depending on the kind of establishment when it came to the selection of risk factors with low importance.

Table 6. Distribution of high priority risk factors selected.

SDLC Process Stages	Baseline Risks	Selected Risks	Ratio Selected (%)
Requirement Analysis and Definition Phase	21	11	52.38
Design Phase	19	8	42.11
Implementation and Unit Testing Phase	23	9	39.13
Integration and System Testing Phase	16	4	25.00
Operation and Maintenance Phase	14	6	42.86

The selection of pertinent risk factors taken into account throughout the SDLC design phase demonstrated that 8 of the 19 risk variables that were previously selected as high priorities, 6 were deemed to be of low priority, and 5 were labeled as risk factors with no priority. The findings also revealed a statistically significant variation in the diverse array of responses according to the type of establishment that they belong to. However, statistically no significant difference in the interpretation based on area of expertise of the respondents that fill the questionnaire.

The results of choosing the pertinent risk factors taken into account during the SDLC's for execution and unit testing stage revealed that 9 of the 23 risk variables that were originally discovered were deemed to be of high priority, 3 risk factors were categorized as having no priority, while 11 was given a poor priority rating. The findings also revealed a significant discrepancy statistically based on the organization the respondents are working for. However, there was no noteworthy change statistically in the developers' varying responses based on their area of expertise.

The outputs of choosing the pertinent risk factors taken into account during the SDLC's system testing and amalgamation stage demonstrated that 4 of the 16 risk factors that were originally selected as high priorities, 1 risk factor was categorized as having no priority, while 11 were low priority. The findings also revealed a statistically significant variation in the diverse array of responses according to the type of establishment that they belong to. However, statistically no significant difference in the interpretation based on area of expertise of the respondents that fill the questionnaire.

The outputs of choosing the pertinent risk factors that were taken into account during the set-up and maintenance stage of the SDLC demonstrated that out of the 14 baseline risk factors, 6 were deemed to be of high priority, while 8 were deemed to be of low priority, and none were designated as risk factors with no priority. The findings further demonstrated that the response diversity did not statistically significantly, depending on the developers' area of expertise. However, there was no noteworthy change statistically in the developers' varying responses based on their area of expertise.

4.2. Identification of Risk Factors of SDLC Phases

The risk factor taxonomy put forward by authors in [5] helped identify a number of risk factors at first. Software developers chosen from various firms recognized a number of risk variables as being pertinent in the context of software development. The analysis of the software engineers' responses led to the following three classifications, thus, the three types of risk factors are: high priority risk factors, which described the risks that all developers agreed upon, low priority risk factors, which described the risks that some developers agreed upon, and no priority risk factors, which defined the risks that no developers agreed upon. Finally, the relevant risk factors for assessing security risk at each SDLC step were determined to be the high-priority risk factors that all the software developers had agreed upon. Figure 4 show the methodology processes.

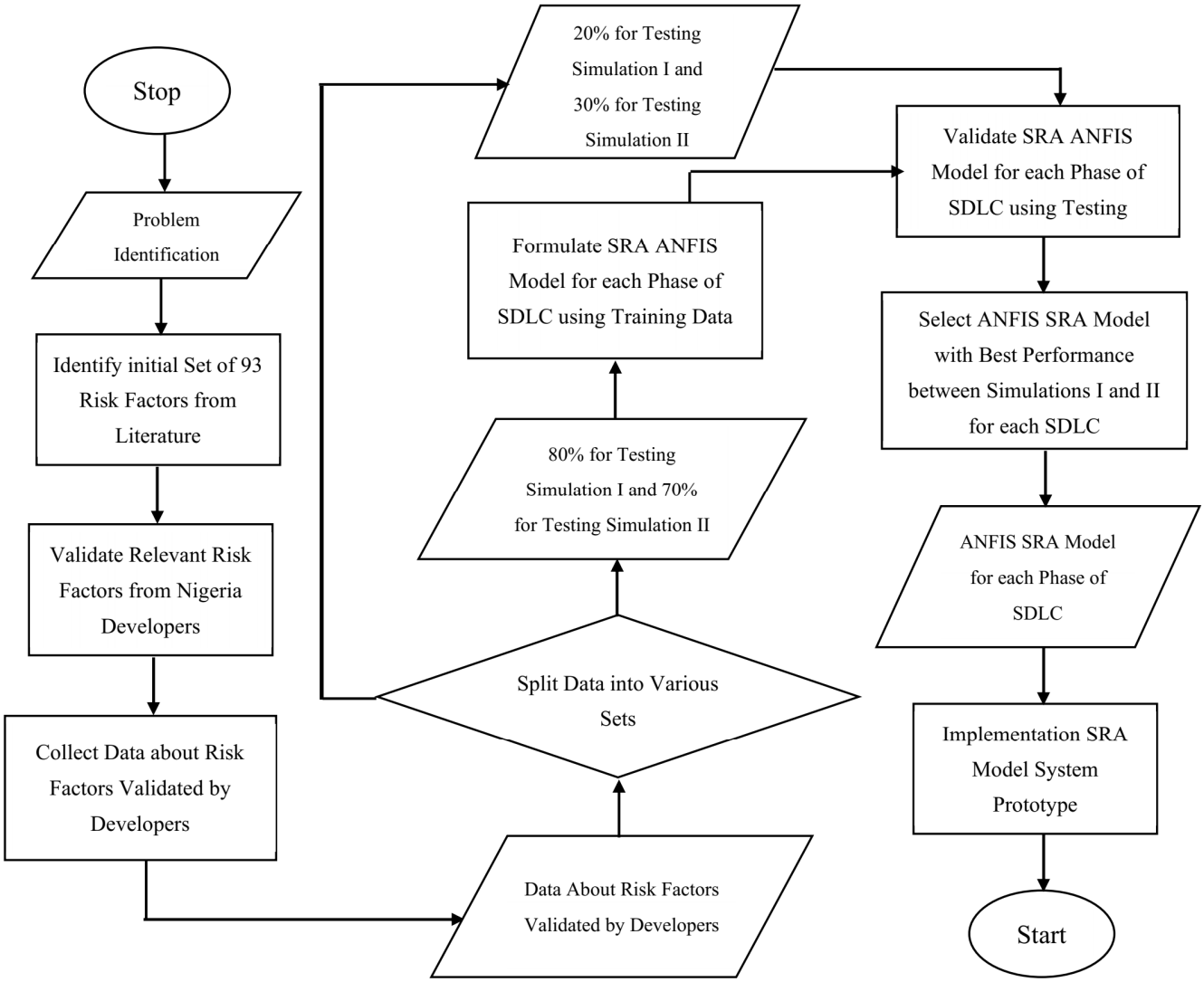

Figure 4. Research Methodology Process.

4.3. The Results of the Proposed ANFIS Enabled with SRA Model

The results of the ANFIS model is presented in this section in accordance with the chosen risk factors in the preceding section. For each SDLC step that was identified, an SDLC security risk assessment model was created based on the proposed model. The proposed model first fuzzify each risk factor before processing it. Figure 5 depicts the ANFIS model's framework utilized the established SRA based on the factors identified in each SDLC process phase. The information flow from the point of input is depicted in the diagram (at the bottom) when the SRA for each stage was established fed the ANFIS model at the point of output (at the top).

The total amount of the user-provided information was calculated to established the each SRA for the SDLC phase. The greater the summation value, the greater the security risk as 1 was applied each time a user responded number that was equivalent to 1. The quantity of risk factors taken into account led to the highest sum being computed, afterwards split into three segments that serve as the threshold as indicated in Equation (1), however, the no-risk class is assigned 0 value. As a percentage of the phase output's overall values, an interval was created using the various components.

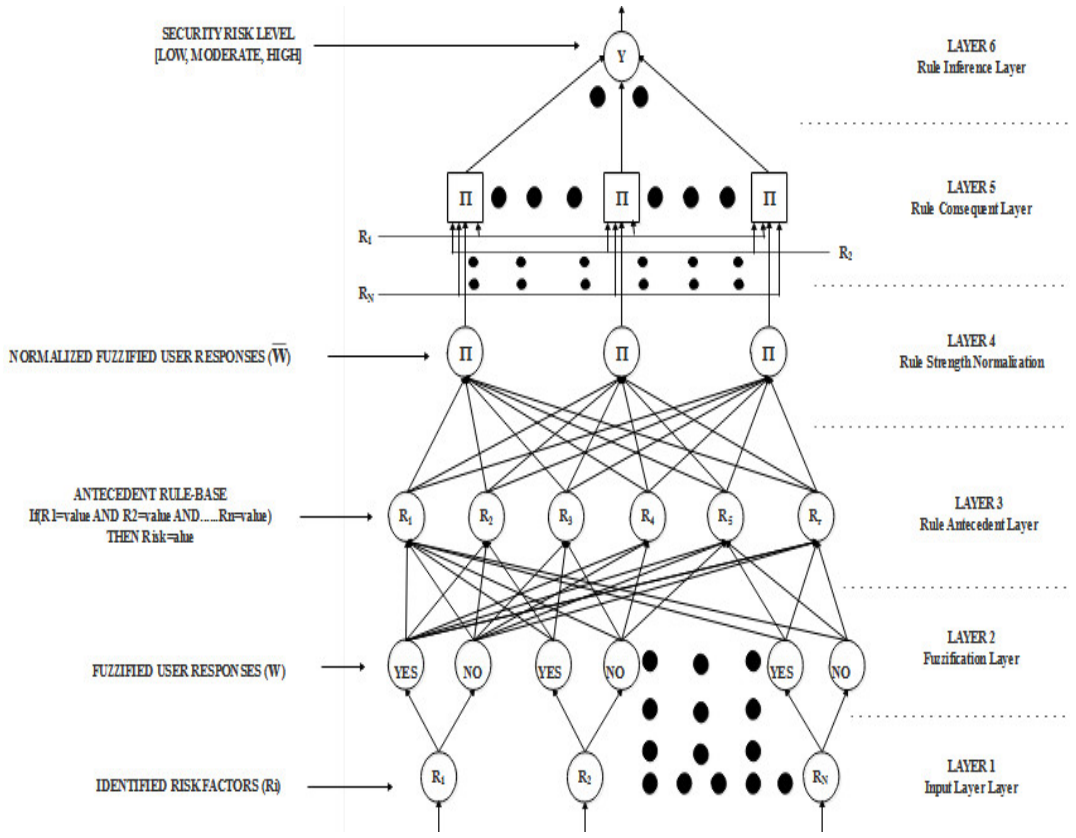

Figure 5. The proposed framework for SRA in software development stages.

For the risk assessment, four triangle membership functions with centers of 0, 1, 2, and 3 were developed for No, Low, Moderate and High-risk classes, respectively. The models were formulated such that a triangular membership function of the interval [−0.5, 0, 0.5] was assigned to the No risk class (Equation (4)), an interval of [0.5, 1, 1.5] were also assigned to the Low risk (Equation (5)), an interval of [1.5, 2, 2.5] were assigned to the Moderate risk (Equation (6)) while an interval of [2.5, 3, 3.5] was then assigned to the High-risk class (Equation (7)). The set of values used to allocate the security risk assessment variables was based on the amount of risk variables taken into account throughout each SDLC phase and distributed as shown in Table 7.

Table 7. Description of Fuzzy and Crisp intervals of ANFIS security risk assessment labels.

SDLC Process Stages	Risk Factors	No [−0.5, 0, 0.5]	Low [0.5, 1, 1.5]	Moderate [1.5, 2, 2.5]	High [2.5, 3, 3.5]
Requirement Analysis and Definition Phase	11	0	[1.4]	[5.8]	[9.11]
Design Phase	8	0	[1.3]	[4.6]	[7.8]
Implementation and Unit Testing Phase	9	0	[1.3]	[4.6]	[7.9]
Integration and System Testing Phase	4	0	[1.2]	3	4
Operation and Maintenance Phase	6	0	[1.2]	[3.4]	[5.6]

The requirement analysis and definition phase's SRA report revealed that 11 high-priority risk factors had been chosen. By assigning a value of 0 to No risk, the total value of 11 was divided between the Low, Moderate, and High-Risk classes. Therefore, no risk was assigned a linguistic value of 0, low risk was assigned a linguistic interval of [1.4], the moderate risk was assigned a linguistic interval of [5.8], and high risk was assigned a

linguistic interval of [9.11]. Therefore, the linguistic value of 0 was assigned a fuzzy interval of [−0.5, 0, 0.5] for no risk, the linguistic interval of [1.4] was assigned a fuzzy interval of [0.5, 1, 1.5] for low risk, the linguistic interval of [5.8] was assigned a fuzzy interval of [1.5, 2, 2.5] for moderate risk while the linguistic interval of [9.11] was assigned a fuzzy interval of [2.5, 3, 3.5] for high risk.

The design phase's SRA result revealed that 8 high priority risk factors had been chosen. By assigning a value of 0 to No risk, the total value of 8 was divided between the Low, Moderate, and High Risk classes. Therefore, no risk was assigned a linguistic value of 0, low risk was assigned a linguistic interval of [1.3], the moderate risk was assigned a linguistic interval of [4.6], and high risk was assigned a linguistic interval of [7.8]. Therefore, the linguistic value of 0 was assigned a fuzzy interval of [−0.5, 0, 0.5] for no risk, the linguistic interval of [1.3] was given a fuzzy interval of [0.5, 1, 1.5] for low risk, the linguistic interval of [4.6] was assigned a fuzzy interval of [1.5, 2, 2.5] for moderate risk while the linguistic interval of [7.8] was assigned a fuzzy interval of [2.5, 3, 3.5] for high risk.

The implementation and unit testing phase's SRA result revealed that nine high-priority risk factors had been chosen. By assigning a value of 0 to No risk, the total value of 8 was divided between the Low, Moderate, and High-Risk classes. Therefore, no risk was assigned a linguistic value of 0, low risk was assigned a linguistic interval of [1.3], the moderate risk was assigned a linguistic interval of [4.6], and high risk was assigned a linguistic interval of [7.9]. Therefore, the linguistic value of 0 was assigned a fuzzy interval of [−0.5, 0, 0.5] for no risk, the linguistic interval of [1.3] was assigned a fuzzy interval of [0.5, 1, 1.5] for low risk, the linguistic interval of [4.6] was assigned a fuzzy interval of [1.5, 2, 2.5] for moderate risk. In contrast, the linguistic interval of [7.9] was assigned a fuzzy interval of [2.5, 3, 3.5] for high risk.

The output from the SRA during the system integration and testing phase indicated that 4 high-priority risk variables had been chosen. By assigning a value of 0 to No risk, the total value of 4 was divided between the Low, Moderate, and High-Risk classes. Therefore, no risk was assigned a linguistic value of 0, low risk was assigned a linguistic interval of [1.2], moderate risk was assigned a linguistic value of 3 and high risk was assigned a linguistic value of 4. Therefore, the linguistic value of 0 was assigned a fuzzy interval of [−0.5, 0, 0.5] for no risk, the linguistic interval of [1.2] was assigned a fuzzy interval of [0.5, 1, 1.5] for low risk, the linguistic value of 3 was assigned a fuzzy interval of [1.5, 2, 2.5] for moderate risk while the linguistic value of 4 was assigned a fuzzy interval of [2.5, 3, 3.5] for high risk.

The operation and maintenance stage results revealed 6 high-priority risk variables had been chosen. By assigning a value of 0 to No risk, the total value of 6 was divided between the Low, Moderate, and High-Risk classes. Therefore, no risk was assigned a linguistic value of 0, low risk was assigned a linguistic interval of [1.2], the moderate risk was assigned a linguistic interval [3.4], and high risk was assigned a linguistic interval of [5.6]. Therefore, the linguistic value of 0 was assigned a fuzzy interval of [−0.5, 0, 0.5] for no risk, the linguistic interval of [1.2] was assigned a fuzzy interval of [0.5, 1, 1.5] for low risk, the linguistic interval of [3.4] was assigned a fuzzy interval of [1.5, 2, 2.5] for moderate risk. In contrast, the linguistic interval of [5.6] was assigned a fuzzy interval of [2.5, 3, 3.5] for high risk.

4.4. Results of the Inference Rules Generated for the ANFIS SRA Model

The findings of the output security risk assessment and the creation of the risk factors in each process demonstrated that number of risk factors detected had the same input variables as the SRA model developed for each phase. To produce an output function required for assessing the security risk from the formulated risk factors, a set of IF-THEN rules which combined the risk factors for determining their respective output was formulated.

The set of rules required for a set of binary-valued (Yes and No) variables is determined by Equation (8) which is expressed in the form expressed in Equation (9).

$$If(RiskFactor1 = value)AND(RiskFactor2).....THEN(SRA = value) \quad (8)$$

where: $value = \{Yes,\ No\}$

$$w_r = \min(\mu_{RF1},\ \mu_{RF2}, \mu_{RF3}, \mu_{RF4}, \mu_{RF5}, \ldots . \mu_{RFi},) \quad (9)$$

where: μ_{RFi} is the fuzzified value of the value provided for each risk factor i; and w_r is the weight of the rule r

Table 8 shows the distribution of the number of inferred rules for each of the 5 ANFIS models used to formulate the model for the SRA of each phase of SDLC. The output of the inferred rules established for the security risk assessment is estimated using Equation (10) which is used to estimate the weighted value, $\overline{w_r}$ for each of the r rules calculated using Equation (8) based on the values of the r rules inferred. The source code for developing the initial fuzzy logic model was formulated to formulate the SRA model for each phase of the SDLC.

$$\overline{w_r} = \frac{w_r}{\sum_{j=1}^{r} w_i} \quad (10)$$

Table 8. Distribution of the number of inferred rules.

The Phase of SDLC Cycle	Selected Risk Factors	Number of Rules
Requirement Analysis and Definition Phase	11	2048
Design Phase	8	256
Implementation and Unit Testing Phase	9	512
Integration and System Testing Phase	4	16
Operation and Maintenance Phase	6	64

4.5. Results of Historical Data Collected Using Follow-Up Questionnaire

Following the evaluation of the risk variables' validity, we observed that they are required to assess the security risk assessment of the different phases of the system development life cycle (SDLC). The follow-up questionnaires distributed across 309 software developers selected from various IT establishments and with varying specialties and age groups are presented. Table 9 shows the distribution of the follow-up questionnaires across the respondents selected based on the establishments they belonged to.

Table 9. Distribution of respondents based on their IT establishments.

Establishments	Frequency	Percentage (%)
General	200	64.72
Nigerian Computer Society (NCS)	10	3.24
APTECH	15	4.85
Tai Solarin University of Education (TASUED)	14	4.53
CC-HUB	11	3.56
Obafemi Awolowo University (IFE)	3	0.97
PyCOM-NG	28	9.06
SIDMACH	28	9.06
Total	309	100.00

Following the collection of the questionnaire distributed for this study, it was observed that the majority were distributed among respondents selected from a number of sources who refused to be disclosed and were referred to as general, with a proportion of 64.7%. This was followed by respondents selected from PyCOM-NG and SIDMACH with a proportion of 9.1% each, APTECH with a proportion of 4.9%, TASUED with a proportion of 4.5%, CC-HUB with a proportion of 3.6%, NCS with a proportion of 3.2% and IFE with a

proportion of 0.97%. Following the respondents' responses regarding each set of risk factors that were used to assess the SRA of each phase of the SDLC process, some observations were made about the responses of the respondents selected from each establishment. Also, the responses made were used to assess the level of security risk based on the number of responses for Yes and No provided by the respondents.

It was observed in the requirements phase results that the majority of the assessments of the responses from APTECH, NCS, PyCOM-NG, and TASUED had no security risk while the majority of the assessments of the responses from CCHUB, IFE, General, and SIDMACH had low-security risk. However, some of the assessments of the responses from CC-HUB, General, IFE, PyCOM-NG, and SIDMACH had a moderate security risk. The results of the design phase showed that the majority of the assessment of the responses from APTECH, NCS, PyCOM-NG, and TASUED had no security risk, while the majority of the assessments of the responses from CC-HUB, General, IFE, and SIDMACH had a low-security risk. However, some of the assessments of the responses from General, PyCOM-NG, and SIDMACH had a moderate security risk. Also, high risk was assessed from a respondent each among General, PyCOM-NG, and SIDMACH respondents.

The results of the implementation phase revealed that the majority of the assessment of the responses from APTECH, NCS, PyCOM-NG, and TASUED had no security risk, while the majority of the assessments of the responses from CCHUB, General, IFE, and SIDMACH had a low-security risk. However, some of the assessments of CC-HUB, General, IFE, PyCOM-NG, and SIDMACH had moderate security risk, while the high-security risk was assessed from a respondent in PyCOM-NG. The results of the integration phase revealed majority of the assessment of the responses from APTECH, CC-HUB, General, NCS, PyCOM-NG, SIDMACH, and TASUED had no security risk. In contrast, the majority of the assessments of the responses from IFE had a low-security risk. However, some of the assessments of the responses from CCHUB, General, PyCOM-NG, and SIDMACH had moderate security risk while the high-security risk was assessed from respondents in PyCOM-NG and SIDMACH.

The results of the operation phase indicated that the majority of the assessment of the responses from APTECH, IFE, NCS, PyCOM-NG, SIDMACH, and TASUED had no security risk while the majority of the assessments of the responses from General low-security risk and an equal majority of the assessments of responses from CCHUB had None and Low-security risks. However, some of the assessments of the responses from CCHUB, General, IFE, PyCOM-NG, SIDMACH, and TASUED had moderate security risk while the high-security risk was assessed from respondents among General and PyCOM-NG.

5. Discussion

This study aimed to identify the risk factors connected with each system development life cycle phase's security evaluation (SDLC). On the basis of an analysis of comparable works, the study first identified a number of risk variables. To determine the most significant risk factors for determining the ASR at each stage in the process software development, questionnaires were given to developers based on these risk factors. After the questionnaires were distributed, the developers identified certain risk indicators, and based on their response, every risk factor was given a priority. Prioritization was given to the risk factors as high, medium, and low. The 100% of respondents approved the use of the identified risk factors, the high significance risk considerations were taken into consideration. The statistical non-parametric test was used to investigate the mid-priority risk factors and look at the variation in replies based on the setting that each respondent belonged to, and the areas of expertise of the chosen respondents. Since the developers did not choose any of the low-priority risk categories, they were all not taken into account.

The results of the study showed that about 93 initial risk factors were initially identified. Among these, 11 risk factors were identified and validated for the requirement analysis and problem identification phase, 8 risk factors were identified and validated for the system design phase, 9 risk factors were identified for the implementation and unit testing phase,

4 risk factors were identified and verified during the phase of system integration and testing, while 6 risk factors were found been approved for the SDLC's operation and maintenance phase. The results showed that using a value of 0 for Yes and 1 for No, a scoring mechanism was created for each phase of the SDLC and was used to determine the security risk classes to be evaluated based on the total score of responses provided by the user. Using the total score, a specific interval was created for Low Risk, Moderate Risk, and High Risk while allocating a score of 0 for No risk cases.

The results showed that using a triangular membership function, and the SRA model was formulated by the use of two triangle membership functions with centers of 0 and 1 for each risk factor's corresponding Yes and No answers. while for the output of the security risk assessment, four triangle membership functions with centers of 0, 1, 2, and 3 were used for No risk, Low risk, Moderate risk, and High risk respectively. The results of the distribution of the security risk assessments made by the respondents showed some observations. Among all the phases of SDLC assessed, it was observed that the requirements phase was the only phase without high security risk assessed however it was observed that the assessment of responses from General, SIDMACH and PyCOM-NG revealed responses that had high security risk following the assessments.

This study shows how proposed model can be used to significantly identify the risk factors in SDLC phases and optimize the prediction of quality attributes in software. To be able to pinpoint the risks relevant to each stage of the SDLC, the risk factors that had been gathered were put to the test in each phase. Findings revealed that identify the risk factors connected to each SDLC phase's security evaluation can increase the software products quality, reliability, portability and usability. Identify the software development risk factor will also increase the software products reliability, functionality, efficiency and maintainability influences that will affect the overall software quality performance most. Hence, users of software systems should place importance to risk factors identity in each stage of SDLC of software products quality attributes based on how relevant the attributes are for the success of the organization. Software developers ought to employ test driven approaches during software development in order to minimize error. Using a developing country to get suggestions on how to encourage more consideration of security throughout the SDLC, has enabled the experts to give ideas that are mostly useful to identify related risk factors. This is opposed to recommendations for specific software development models, to accomplish the necessary increase in attention to the identified risk factors in each SDLC stage for software products security. Security-related SDLC models are beneficial if the software development team members apply them. Although some panelists addressed techniques, no particular process models were put forth.

6. Conclusions

Risk assessments are performed for a variety of reasons. The risk of software-based systems is reduced through the incorporation of risk assessment processes. In many organizations, including security in every stage of the SDLC is a laborious task. Therefore, this study suggests a new ANFIS-based method for carrying out risk analysis during the software development process. The study was able to reduce the number of risk factors that were recommended by related works to a set recommended risk factors suitable for assessing security risk in the Nigerian context in the development of software product. It was concluded from the study that among the initially identified risk factors for the SDLC phases, 11, 8, 9, 4, and 6 risk factors were validated by system developers for the assessment of each SDLC phase from the requirement to the operation and maintenance phase respectively. The study also concluded that by allocating values of 0 and 1 to Yes and No responses provided by the responses of stakeholders to questions about risk factors considered at each stage, the total score was split into 4 parts which provided a means of assessing risk as either of No, Low, Moderate and High Risk for each phase of SDLC. It was concluded from the study that using this crude scoring system, the dataset that was required for the formulation of the SRA model for the SDLC phases was developed

by adopting 2 and 4 Fuzzy triangular membership functions for each input risk factors and output SRA class respectively. An identification of the effect of each risk factor on the security risk posed to each phase can be used to create a weighting scheme of ensuring the development of an effective scoring system of security risk assessment in SDLC process, and this will be explored in our future work.

Author Contributions: The manuscript was written through the contributions of all authors. Conceptualization, R.G.J. and O.O.O.; methodology, R.G.J. and O.O.O.; software, O.O.O. and J.B.A.; validation, A.L.I., C.-C.L. and O.O.O.; formal analysis, A.L.I.; investigation, J.B.A.; resources, R.G.J.; data curation, O.O.O.; writing—original draft preparation, J.B.A.; writing—review and editing, R.G.J., O.O.O., C.-C.L. and A.L.I.; visualization, J.B.A. and C.-C.L.; supervision, J.B.A.; project administration, R.G.J., O.O.O. and A.L.I.; funding acquisition, J.B.A. and A.L.I. All authors have read and agreed to the published version of the manuscript.

Funding: The work of Agbotiname Lucky Imoize is supported in part by the Nigerian Petroleum Technology Development Fund (PTDF) and in part by the German Academic Exchange Service (DAAD) through the Nigerian-German Postgraduate Program under grant 57473408.

Data Availability Statement: The data that support the findings of this paper are available upon reasonable request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sahu, K.; Alzahrani, F.A.; Srivastava, R.K.; Kumar, R. Hesitant fuzzy sets based symmetrical model of decision-making for estimating the durability of Web application. *Symmetry* **2020**, *12*, 1770. [CrossRef]
2. Islam, G.; Storer, T. A case study of agile software development for safety-Critical systems projects. *Reliab. Eng. Syst. Saf.* **2020**, *200*, 106954. [CrossRef]
3. Imoize, A.L.; Idowu, D.; Bolaji, T. A brief overview of software reuse and metrics in software engineering. *World Sci. News* **2019**, *122*, 56–70.
4. Awotunde, J.B.; Ayo, F.E.; Ogundokun, R.O.; Matiluko, O.E.; Adeniyi, E.A. Investigating the roles of effective communication among stakeholders in collaborative software development projects. In Proceedings of the International Conference on Computational Science and Its Applications, Cagliari, Italy, 1–4 July 2020; Volume 12254, pp. 311–319.
5. Hijazi, H.; Alqrainy, S.; Muaidi, H.; Khdour, T. Risk Factors in Software Development Phases. *Eur. Sci. J.* **2014**, *10*, 213–232.
6. Sahu, K.; Shree, R.; Kumar, R. Risk management perspective in SDLC. *Int. J. Adv. Res. Comput. Sci. Softw. Eng.* **2014**, *4*, 1247–1251.
7. Awotunde, J.B.; Misra, S.; Adeniyi, A.E.; Abiodun, M.K.; Kaushik, M.; Lawrence, M.O. A Feature Selection-Based K-NN Model for Fast Software Defect Prediction. In Proceedings of the International Conference on Computational Science and Its Applications, Malaga, Spain, 4–7 July 2022; Springer: Cham, Switzerland, 2022; pp. 49–61.
8. Behera, P.C.; Dash, C.; Pareek, P.K. A Novel Approach for Improving Security in Software Development in Small Software Firms: A Literature Review. In *Emerging Technologies in Data Mining and Information Security*; Springer: Singapore, 2021; pp. 689–698.
9. Saputri, T.R.D.; Lee, S.-W. Integrated framework for incorporating sustainability design in software engineering life-cycle: An empirical study. *Inf. Softw. Technol.* **2020**, *129*, 106407. [CrossRef]
10. Unuakhalu, M.F.; Sigdel, D.; Garikapati, M. Integrating Risk Management in System Development Cycle. *Int. J. Softw. Web Sci.* **2014**, *8*, 1–9.
11. Laaraib, E.; Maher, Z.A.; Solangi, Z.A.; Koondhar, M.Y.; Memon, M.; Depar, M.; Shah, A. A Methodology for Incorporating Quality Assurance Practices during Software Development Life Cycle. *Int. J.* **2021**, *10*, 2296–2301.
12. Gandhi, A.; Naik, A.; Thakkar, K.; Gahirwal, M. Risk Management in Software Development using Artificial Neural Networks. *Int. J. Comput. Appl.* **2014**, *93*, 22–27. [CrossRef]
13. Imoize, A.L.; Mekiliuwa, S.C.; Omiogbemi, I.M.B. Recent Trends on the Application of Cost-Effective Economics Principles to Software Engineering Development. *Int. J. Inf. Secur. Softw. Eng.* **2020**, *6*, 39–49.
14. Khan, R.A.; Khan, S.U.; Khan, H.U.; Ilyas, M. Systematic mapping study on security approaches in secure software engineering. *IEEE Access* **2021**, *9*, 19139–19160. [CrossRef]
15. Dodson, D.; Souppaya, M.; Scarfone, K. *Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework (ssdf)*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2020. [CrossRef]
16. Shameem, M.; Kumar, C.; Chandra, B.; Khan, A.A. Systematic review of success factors for scaling agile methods in global software development environment: A client-vendor perspective. In Proceedings of the 2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW), Nanjing, China, 4–8 December 2017; IEEE: New York, NY, USA, 2017; pp. 17–24.
17. Vochitoiu, H.; Vedinas, F.; Miclea, O.; Unguras, C.L. Risk Management as a Part of the Business Process in Corporate Firms. In Proceedings of the International Conference "New Technologies, Development and Applications", Sarajevo, Bosnia and Herzegovina, 25–27 June 2020; Springer: Cham, Switzerland; pp. 964–972.

18. Imoize, A.L.; Mekiliuwa, S.C.; Omiogbemi, I.M.B.; Omofonma, D.O. Ethical Issues and Policies in Software Engineering. *Int. J. Inf. Secur. Softw. Eng.* **2020**, *6*, 6–17.
19. Pooja, R.; Dalwinder, S.S. Neuro-Fuzzy based Software Risk Estimation Tool. *Glob. J. Comput. Sci. Technol. Softw. Data Eng.* **2013**, *13*, 23–34.
20. Casola, V.; De Benedictis, A.; Rak, M.; Villano, U. A novel Security-by-Design methodology: Modeling and assessing security by SLAs with a quantitative approach. *J. Syst. Softw.* **2020**, *163*, 110537. [CrossRef]
21. Hart, S.; Margheri, A.; Paci, F.; Sassone, V. Riskio: A serious game for cyber security awareness and education. *Comput. Secur.* **2020**, *95*, 101827. [CrossRef]
22. Al-Matari, O.M.; Helal, I.M.; Mazen, S.A.; Elhennawy, S. Adopting security maturity model to the organizations' capability model. *Egypt. Inform. J.* **2021**, *22*, 193–199. [CrossRef]
23. Rindell, K.; Bernsmed, K.; Jaatun, M.G. Managing security in software: Or: How I learned to stop worrying and manage the security technical debt. In Proceedings of the 14th International Conference on Availability, Reliability and Security, Canterbury, UK, 26–29 August 2019; pp. 1–8.
24. Nguyen, J.; Dupuis, M. Closing the feedback loop between ux design, software development, security engineering, and operations. In Proceedings of the 20th Annual SIG Conference on Information Technology Education, Tacoma, WA, USA, 3–5 September 2019; pp. 93–98.
25. Jouini, M.; Rabai, L.B.A.; Khedri, R. A quantitative assessment of security risks based on a multifaceted classification approach. *Int. J. Inf. Secur.* **2021**, *20*, 493–510. [CrossRef]
26. Akbar, M.A.; Shameem, M.; Ahmad, J.; Maqbool, A.; Abbas, K. Investigation of Project Administration related challenging factors of Requirements Change Management in global software development: A systematic literature review. In Proceedings of the 2018 International Conference on Computing, Electronic and Electrical Engineering (ICE Cube), Quetta, Pakistan, 12–13 November 2018; IEEE: New York, NY, USA, 2018; pp. 1–7.
27. Podari, Z.; Arbain, A.F.; Ibrahim, N.; Sudarmilah, E. Risk Mitigation Framework for Agile Global Software Development. In Proceedings of the International Conference of Reliable Information and Communication Technology, Langkawi, Malaysia, 21–22 December 2020; Springer: Cham, Switzerland, 2020; pp. 1233–1246.
28. Wong, W.E.; Li, X.; Laplante, P.A. Be more familiar with our enemies and pave the way forward: A review of the roles bugs played in software failures. *J. Syst. Softw.* **2017**, *133*, 68–94. [CrossRef]
29. Akinsola, J.E.; Ogunbanwo, A.S.; Okesola, O.J.; Odun-Ayo, I.J.; Ayegbusi, F.D.; Adebiyi, A.A. Comparative analysis of software development life cycle models (SDLC). In Proceedings of the Computer Science On-line Conference, Jeju, Korea, 14–16 July 2020; Springer: Cham, Switzerland, 2020; pp. 310–322.
30. Magableh, A.A.; Alsobeh, A.M.R. Aspect-Oriented Software Security Development Life Cycle (AOSSDLC). In Proceedings of the CS & IT Conference Proceedings, Dubai, United Arab Emirates, 25–26 August 2018.
31. Agarwal, P.; Singhal, A.; Garg, A. SDLC Model Selection Tool and Risk Incorporation. *Int. J. Comput. Appl.* **2017**, *975*, 8887. [CrossRef]
32. Khan, M.N.A.; Mirza, A.M.; Saleem, I. Software Risk Analysis with the use of Classification Techniques: A Review. *Eng. Technol. Appl. Sci. Res.* **2020**, *10*, 5678–5682. [CrossRef]
33. Mohammad, A.; Alqatawna, J.F.; Abushariah, M. Secure software engineering: Evaluation of emerging trends. In Proceedings of the 2017 8th International Conference on Information Technology (ICIT), Amman, Jordan, 17–18 May 2017; IEEE: New York, NY, USA, 2017; pp. 814–818.
34. Sharif, A.M.; Basri, S.; Ali, H.O. Strength and Weakness of Software Risk Assessment Tools. *Int. J. Softw. Eng. Its Appl.* **2014**, *8*, 389–398.
35. Abioye, T.E.; Arogundade, O.T.; Misra, S.; Akinwale, A.T.; Adeniran, O.J. Toward ontology-based risk management framework for software projects: An empirical study. *J. Softw. Evol. Process* **2020**, *32*, e2269. [CrossRef]
36. Jackson, A.B.; Jackson, T.; Jackson, K.B. Chronology of continuous improvement of the world's best FMECA standard. In Proceedings of the 2020 Annual Reliability and Maintainability Symposium (RAMS), Palm Springs, CA, USA, 17–30 January 2020; IEEE: New York, NY, USA, 2020; pp. 1–6.
37. Scheu, M.N.; Tremps, L.; Smolka, U.; Kolios, A.; Brennan, F. A systematic Failure Mode Effects and Criticality Analysis for offshore wind turbine systems towards integrated condition based maintenance strategies. *Ocean. Eng.* **2019**, *176*, 118–133. [CrossRef]
38. Androulidakis, I.; Kharchenko, V.; Kovalenko, A. Imeca-based technique for security assessment of private communications: Technology and training. *Inf. Secur.* **2016**, *35*, 99. [CrossRef]
39. Babeshko, I.; Illiashenko, O.; Kharchenko, V.; Leontiev, K. Towards Trustworthy Safety Assessment by Providing Expert and Tool-Based XMECA Techniques. *Mathematics* **2022**, *10*, 2297. [CrossRef]
40. Oleg, I.; Vyacheslav, K.; Andriy, K. Cyber security lifecycle and assessment technique for FPGA-based I & C systems. In Proceedings of the East-West Design & Test Symposium (EWDTS 2013), Rostov on Don, Russia, 27–30 September 2013; IEEE: New York, NY, USA, 2013; pp. 1–5.
41. Kumar, R.; Schivo, S.; Ruijters, E.; Yildiz, B.M.; Huistra, D.; Brandt, J.; Stoelinga, M. Effective analysis of attack trees: A model-driven approach. In Proceedings of the International Conference on Fundamental Approaches to Software Engineering, Thessaloniki, Greece, 14–21 April 2018; Springer: Cham, Switzerland, 2018; pp. 56–73.

42. Lallie, H.S.; Debattista, K.; Bal, J. A review of attack graph and attack tree visual syntax in cyber security. *Comput. Sci. Rev.* **2020**, *35*, 100219. [CrossRef]
43. Mutlu, B.O.; Kestor, G.; Manzano, J.; Unsal, O.; Chatterjee, S.; Krishnamoorthy, S. Characterization of the impact of soft errors on iterative methods. In Proceedings of the 2018 IEEE 25th International Conference on High Performance Computing (HiPC), Bengaluru, India, 17–20 December 2018; IEEE: New York, NY, USA, 2018; pp. 203–214.
44. Chen, J.; Li, Q.; Mao, C.; Towey, D.; Zhan, Y.; Wang, H. A web services vulnerability testing approach based on combinatorial mutation and soap message mutation. *Serv. Oriented Comput. Appl.* **2014**, *8*, 1–13. [CrossRef]
45. Schoenfield, B.; Ransome, J.; Misra, A. Applying the SDL Framework to the Real World. In *Core Software Security: Security at the Source*; CRC Press: Boca Raton, FL, USA, 2014; pp. 255–324.
46. Gonzalez, D. The State of Practice for Security Unit Testing: Towards Data Driven Strategies to Shift Security into Developer's Automated Testing Workflows. Ph.D. Thesis, Rochester Institute of Technology, Rochester, NY, USA, 2021.
47. Masso, J.; Pino, F.J.; Pardo, C.; García, F.; Piattini, M. Risk management in the software life cycle: A systematic literature review. *Comput. Stand. Interfaces* **2020**, *71*, 103431. [CrossRef]
48. Liu, D.; Wang, Q.; Xiao, J. The role of software process simulation modeling in software risk management: A systematic review. In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, Washington, DC, USA, 15–16 October 2009; IEEE: New York, NY, USA, 2009; pp. 302–311.
49. Choetkiertikul, M.; Sunetnanta, T. A risk assessment tool using a CMMI Quantitative Approach. *IACSIT Int. J. Eng. Technol.* **2012**, *4*, 352–353. [CrossRef]
50. Iranmanesh, S.H.; Khodadadi, S.B.; Taheri, S. Risk Assessment of Software Projects Using Fuzzy Interface System. In Proceedings of the International Conference on Computing and Industrial Engineering (CIE), Troyes, France, 6–9 July 2009; pp. 1149–1154.
51. Ansari, M.T.J.; Pandey, D.; Alenezi, M. STORE: Security threat oriented requirements engineering methodology. *J. King Saud Univ.—Comput. Inf. Sciences* **2018**, *34*, 191–203. [CrossRef]
52. Alenezi, M.; Almuairfi, S. Security risks in the software development lifecycle. *Int. J. Recent Technol. Eng.* **2019**, *8*, 7048–7055. [CrossRef]
53. Barabanov, A.V.; Markov, A.S.; Grishin, M.I.; Tsirlov, V.L. Current taxonomy of information security threats in software development life cycle. In Proceedings of the 2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT), Almaty, Kazakhstan, 17–19 October 2018; IEEE: New York, NY, USA, 2018; pp. 1–6.
54. Mohino, J.D.V.; Higuera, B.; Montalvo, J.A.S. The application of a new secure software development life cycle (S-SDLC) with agile methodologies. *Electronics* **2019**, *8*, 1218. [CrossRef]
55. Akbar, M.A.; Sang, J.; Khan, A.A.; Amin, F.E.; Nasrullah; Shafiq, M.; Hussain, S.; Hu, H.; Elahi, M.; Xiang, H. Improving the quality of software development process by introducing a new methodology–AZ-model. *IEEE Access* **2017**, *6*, 4811–4823. [CrossRef]
56. Karim, N.S.A.; Albuolayan, A.; Saba, T.; Rehman, A. The practice of secure software development in SDLC: An investigation through existing model and a case study. *Secur. Commun. Netw.* **2016**, *9*, 5333–5345.
57. Sahu, K.; Alzahrani, F.A.; Srivastava, R.K.; Kumar, R. Evaluating the impact of prediction techniques: Software reliability perspective. *Comput. Mater. Contin.* **2021**, *67*, 1471–1488. [CrossRef]