



Article

An Exploratory Study Gathering Security Requirements for the Software Development Process

Roberto Andrade ¹, Jenny Torres ¹, Iván Ortiz-Garcés ^{2,*}, Jorge Miño ¹ and Luis Almeida ¹

¹ Software Security Anti-patterns Research Group (SSA-RG), Facultad de Ingeniería de Sistemas, Escuela Politécnica Nacional, Quito 170525, Ecuador; roberto.andrade@epn.edu.ec (R.A.); jenny.torres@epn.edu.ec (J.T.); jorge.mino@epn.edu.ec (J.M.); luis.almeida01@epn.edu.ec (L.A.)

² Escuela de Ingeniería en Ciberseguridad, Facultad de Ingeniería y Ciencias Aplicadas, Universidad de Las Américas, Quito 170125, Ecuador

* Correspondence: ivan.ortiz@udla.edu.ec

Abstract: Software development stands out as one of the most rapidly expanding markets due to its pivotal role in crafting applications across diverse sectors like healthcare, transportation, and finance. Nevertheless, the sphere of cybersecurity has also undergone substantial growth, underscoring the escalating significance of software security. Despite the existence of different secure development frameworks, the persistence of vulnerabilities or software errors remains, providing potential exploitation opportunities for malicious actors. One pivotal contributor to subpar security quality within software lies in the neglect of cybersecurity requirements during the initial phases of software development. In this context, the focal aim of this study is to analyze the importance of integrating security modeling by software developers into the elicitation processes facilitated through the utilization of abuse stories. To this end, the study endeavors to introduce a comprehensive and generic model for a secure software development process. This model inherently encompasses critical elements such as new technologies, human factors, and the management of security for the formulation of abuse stories and their integration within Agile methodological processes.

Keywords: software security; security software process; security software methodologies; security testing



Citation: Andrade, R.; Torres, J.; Ortiz-Garcés, I.; Miño, J.; Almeida, L. An Exploratory Study Gathering Security Requirements for the Software Development Process. *Electronics* **2023**, *12*, 3594. <https://doi.org/10.3390/electronics12173594>

Academic Editor: Scott Uk-Jin Lee

Received: 22 May 2023

Revised: 14 August 2023

Accepted: 18 August 2023

Published: 25 August 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In the contemporary landscape, businesses embrace digital transformation to enhance productivity and optimize their strategic processes. This metamorphosis necessitates incorporating various types of software (operational, strategic, and business-oriented). Nonetheless, organizations encounter challenges during the software development journey, aiming to culminate in a refined version endowed with quality and security attributes. A report commissioned by Syopsis [1] underscores the substantial financial repercussions stemming from subpar software quality, amounting to nearly USD 2.4 trillion. This report identifies three primary predicaments:

1. Software vulnerabilities;
2. The software supply chain;
3. Technical debt.

In this context, software security is relevant in both the academic and industrial sectors [2]. According to McGraw et al. [3], software security is defined as the concept of engineering software that can consistently function correctly even under malicious attacks. However, from the point of view of Sametinger et al. [4], software security often remains overlooked by organizations, developers, and end-users, and according to Byres et al. [5], security is frequently an afterthought during software development, only being addressed through activities like penetration testing. Despite the possibility of incorporating security

features into each phase of the software development process through the Secure Software Development Life Cycle (S-SDLC) [6], vulnerabilities in software solutions that attackers can exploit may still arise.

According to Islam et al. [2], the issue lies in the immaturity of software security measurement, and methods to provide a complete picture for measuring software security are lacking. Nevertheless, there are several standards, guidelines, methodologies, and certifications related to software security that can guide software developers to build more secure and compliant software projects. This context has given rise to the three research questions present in this study:

1. What components of the software development process cause these vulnerabilities to still exist?
2. Are there shortcomings in the software development process that prevent vulnerabilities from being perceived by software developers?
3. Is there a lack of knowledge about secure development frameworks or tools that could be used?

According to Braz et al. [7], organizations are increasingly prioritizing security in the earlier stages of software development, notably during code review. However, the study proposed by Braz indicates that developers often struggle with identifying security issues in the early phases due to a lack of training and security knowledge. Currently available security training for software development tends to focus on fundamental cybersecurity principles and best practices for writing secure code. Nonetheless, the challenge might run deeper, involving the need for comprehensive training in software security that encompasses the cognitive processes and capabilities of software developers.

Software developers typically approach their work with an optimistic outlook on how the software should function, often overlooking potential malicious uses [8]. Techniques such as misuse cases or abuse stories can be utilized during the software elicitation process to incorporate security aspects, commonly known as non-functional requirements (NFRs). Thomas et al. [9] suggests that the abuse story format enhances the realism of threat modeling, offering insights into how attackers could exploit vulnerabilities and creating a tangible sense of risk. In contrast, NFRs related to security remain too abstract to evoke genuine engagement.

The literature review shows that abuse stories could be an important component of the elicit software process, as they help to identify potential security vulnerabilities and risks associated with the software. Nevertheless, interviews conducted with academics and professionals in the software and cybersecurity domains reveal a lack of extensive knowledge regarding the practical application of abuse stories. Despite certain existing research on abuse stories during the software elicitation process, significant research gaps remain and require attention.

1. Lack of standardization: There is no standard or framework for developing abuse stories in the elicit software process. For organizations, it is not easy to develop abuse stories in a consistent and effective way for the software development process.
2. Limited empirical research: There is a lack of empirical research on the effectiveness of abuse stories for improving software security. While some studies have suggested including abuse stories, the most effective approaches for developing and using abuse stories in the software development process must be determined.
3. Limited consideration of user experience: Many abuse stories focus primarily on security requirements, without considering the user experience in their development.
4. Lack of integration with Agile methodologies: There is a need for additional research on how to integrate abuse stories into Agile methodologies.

Future research could center on the development of novel techniques for eliciting abuse stories while also assessing the efficacy of current methods in pinpointing potential abuse scenarios. Furthermore, a deeper exploration is warranted into the seamless integration of abuse stories within the software development lifecycle, guaranteeing the early

identification and mitigation of potential risks. Considering the burgeoning utilization of emerging technologies like IoT and blockchain, it is paramount to incorporate abuse stories as a preventive measure against security incidents given the significant expansion in their adoption across various domains.

The objective of this study is to analyze the contribution of abuse stories and how they could be used in Agile methodologies through the means of an exploratory study of gathering security requirements on the software development process, which allows addressing the challenges and future work in this area. To achieve this objective, a model for the Software Security Development Process (SSDP) is proposed, structured into four main components:

- (a) Use of software development in new environments;
- (b) Human factors in software development;
- (c) Management of cybersecurity in software development;
- (d) The process of cybersecurity gathering requirements on software development.

Based on the proposal model, the process of using of abuse stories in Agile methodologies is developed. This study is structured as follows: Section 2 presents the methodology used to establish the exploratory analysis related to software security aspects through a Systematic Literature Review (SLR). Section 3 shows the results obtained by the SLR. Finally, Section 4 introduces a discussion on the challenges that must be addressed in the field of software security.

2. Materials and Methods

To model scenarios involving malicious interactions with the software product by bad users or attackers, it is necessary to initially comprehend the challenges of software security. To address this aspect, a literature review of articles related to the topic from the last five years is conducted. The methodology for conducting the SLR is based on PILAR, comprising four phases (refer to Figure 1):

1. Identification included various steps, such as study selection, inclusion and exclusion criteria, manual search, and removal of duplicates.
2. Screening, consisting of the reviewing process of titles and abstracts.
3. Eligibility analysis was conducted by reading the full texts of the selected articles.
4. Inclusion consisted in data extractions.

Study selection was based on a systematic review following the Prisma Guidelines. It was conducted utilizing the following databases: Springer, Scopus, IEEE, Association for Computing Machinery (ACM), Web of Science, and Science Direct. These databases were chosen since they are the most relevant sources of information corresponding to Computer Science. All publications from 2017 to 2022 were included in the research, using the keyword string “(security AND software)” AND “(cybersecurity AND Software)”. The inclusion criteria comprised (i) documents published by peer-reviewed academic sources and (ii) documents that considered the inclusion of methodologies, frameworks, or approaches in any phase of software development cycle. In addition, the exclusion criteria included (i) studies that lack technical details regarding security concepts. Employing the aforementioned keyword strings, a total of 1966 papers related to software security were identified.

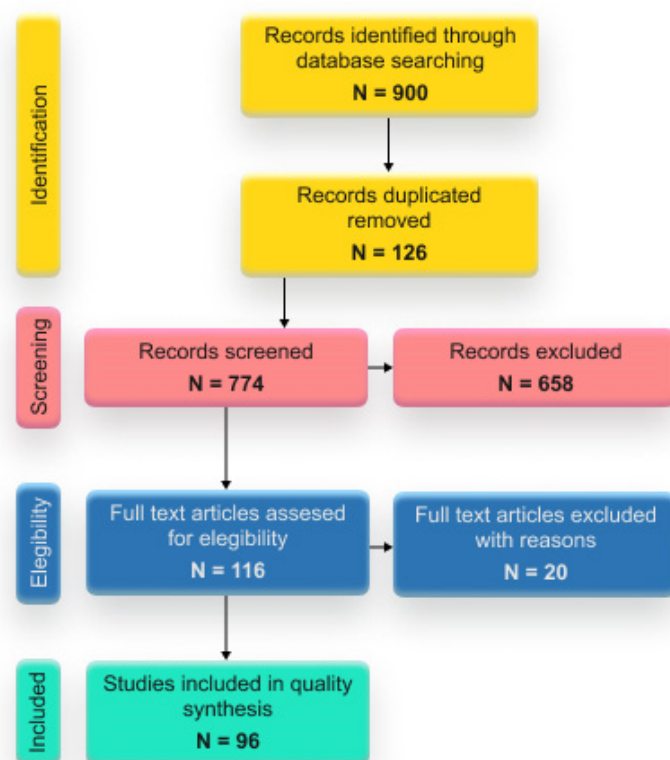


Figure 1. Systematic Literature Review based on Prisma Guidelines.

The screening process of 900 papers after the process of deleting duplicates was conducted based on the review of paper titles and abstracts using a web application created for the systematic review process, Rayyan. The web application allows the examination of the titles and abstracts of the collected papers by each reviewer while maintaining a blinded review process. At the conclusion of the screening process, all reviewers were able to view the papers selected by the other reviewers. All disagreements between reviewers related to some excluded or included articles were resolved. At the end of this step, 126 articles that met the criteria remained in the selected group. In the inclusion stage, the full text of each article was examined for the data extraction stage. For each article selected, the following information was summarized: (i) type of security software methodology, (ii) security software testing, and (iii) security software design process.

From the systematic literature review, a classification of factors requiring consideration in software development is proposed to construct models of cybersecurity aspects that may impact the software product, either positively or negatively. Based on these factors, we propose a model to address the secure software development process and then, from this model, the process is developed to include abuse stories as an element of Agile methodologies.

To validate the proposal, the collaboration of two groups of students from software-related disciplines in their final semester is utilized. The first group of students is taking the course on the construction of software, while the second group is taking the course on software quality assurance. The first group of students builds software in two ways, first without using abuse stories and taking security as a non-functional requirement and then building the software using abuse stories. Meanwhile, the second group of students is focused on validating the security of the product from a software testing perspective. The exercises associated with abuse stories are conducted over one month.

3. Results

3.1. Use of Software Development in New Environments

Based on the literature review, software security is relevant due to technological growth. As illustrated in Figure 2, a thematic cloud of articles was obtained from the Rayyan tool. In this figure, it is possible to observe that software security is considered in the development of the Internet of Things, Software Defined Networks (SDN), machine learning, deep learning, blockchain, and cloud computing.

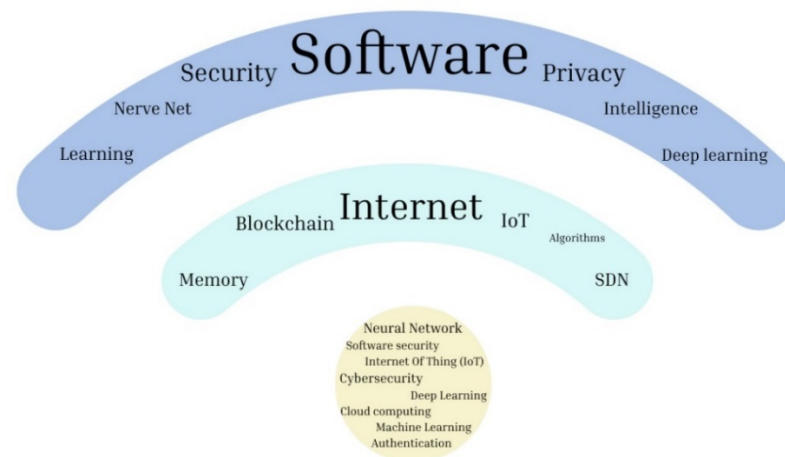


Figure 2. Topics related with software security using Rayyan systematic tool.

(a) Emerging applications

According to the data derived from the literature review, software security is related to emerging applications such as the Internet of Things or blockchain. These technologies have gained relevance in the digital transformation processes and the promotion of Smart concepts [10]. Their applicability lies in different verticals such as health, education, transportation, energy, and science. Some of these verticals may be more critical than others in terms of the type of information or availability of access to resources, such as health or energy, which drives the need to reinforce good security practices.

(b) Internet of things

The Internet of Things (IoT) makes it possible to incorporate intelligence into physical elements such as lights, refrigerators, and SCADA systems by connecting them to the Internet. IoT devices are computers, with a specifically designed operating system and supporting different programming languages such as Python, Java, and C++, among others, so they essentially have the same problems related to software security aspects as in information systems, secure device authentication methods and encryption of sending information. Bagchi et al. [11] mention the importance of software security in IoT devices considering the growth of IoT. According to Bagchi, there are 9 billion embedded processors, outnumbering the human population.

(c) Blockchain

Digital transformation processes have increased the need for integration between information systems or applications. This type of integration requires the establishment of strong authentication mechanisms to prevent attacks such as information or identity theft. PKI-based architectures have been considered to solve these security issues, but some environments may find these infrastructures costly to maintain [12]. In this regard, blockchain-based architectures have been considered to solve authentication challenges. In essence, the agreement between parties is based on smart contracts, but like any software program, smart contracts are susceptible to contain security vulnerabilities [13].

(d) Software-Defined Networks (SDNs)

SDNs have experienced significant growth in programmability, traffic management, and adaptive configuration. Nevertheless, they introduce certain challenges related to privacy, as mentioned by Horvath [14]. Attacks can be directed toward either the data plane or the controller layer of an SDN. Issues related to packet sniffing could be present in an SDN due to data sharing between devices [15].

3.2. The Human Factors on Software Development

The software development process directly or indirectly involves various human factors. Figure 3 summarizes aspects of the process of software development that indirectly associates the use of human factors for its accomplishment identified from the literature review.

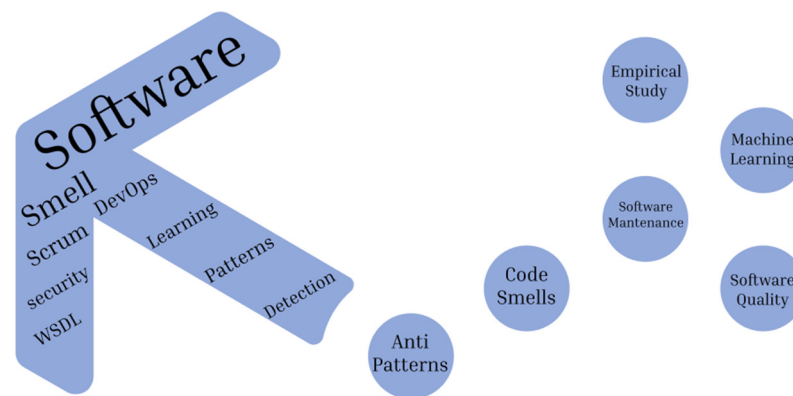


Figure 3. Topics related with human factor on software security using Rayyan systematic tool.

One cognition process of the construction of software development is abstraction through this process is possible to model the behavior of the system, the interaction of the user, and cybersecurity aspects. Another aspect import of abstraction is allowing the optimization of the construction of software. Abstraction makes it possible to hide unnecessary code programming details for other components of the system or for other systems. Having less complex or extensive code makes it easier to determine malicious behavior. Two types of abstraction are Data and Process abstractions. Even the abstraction process must try not to hide too little to maintain the security risk, nor hide too much to conceal functionality from users. However, developers may make mistakes when implementing abstractions due to several factors, some of which include:

- Lack of skills to develop abstraction processes;
- Cognitive biases

Another method that indirectly has human factors is code refactoring, which is the process of restructuring code without changing functionality or behavior. This technique can be used to maintain legacy software. Nonetheless, the problem is that the reused code may have embedded malicious code or some vulnerability due to programming errors.

In the process of software development, two potential problems for cybersecurity might appear: code smell and antipatterns. Antipatterns are design patterns that are counterproductive and ineffective in solving, while code smell is associated with bad practices of software development. Moreover, even the utilization of design patterns could introduce cybersecurity concerns if they are not appropriately configured.

3.3. The Management of Cybersecurity on Software Development

The integration between systems, applications, and the interconnectivity of physical objects to the Internet increase the challenges of cybersecurity. A relevant aspect of societal dynamics is the demand for data sharing. Wu et al. [16] mentions that data sharing increases the risk of information leakage and data loss and considers the use of data encryption important. In the context of high interconnectivity and interdependence and the need

to protect data, developers must take software security into account, regardless of the technology or programming language used. Inger et al. [17] mentions the need to promote periodic meetings to evaluate and make decisions about software security. In addition, Inger mentions that the limitation of Agile software development is that it is not possible to define prescriptive approaches to software security because most sprint evaluation meetings have a retrospective approach to evaluate development progress and milestone achievement. In the same line, Raluca et al. [18] mentions that the origin of software vulnerabilities is the insufficient attention in the software development life cycle. According to Ardagna et al. [19], software process certification is widely recognized to increase system reliability and reduce uncertainty in decision-making. To acquire an adequate level of certification, it is important to follow the approaches defined by some methodologies and frameworks. In the context of software security, the following software assessment methods can be employed:

- OWASP DevSecOps Maturity Model (DSOMM),
- OWASP Software Assurance Maturity Model (SAMM),
- OWASP CLASP (Comprehensive, Lightweight Application Security Process),
- Building Security in Maturity Model (BSIMM),
- Socratic methodology,
- CMMI,
- Microsoft trustworthy Computing SDL,
- SPICE.

Although some methodologies do not focus exclusively on software security, they are important for assessing the risk level of potential threats. The following are some of them:

- OCTAVE allows estimating risk based on the analysis of vulnerabilities and potential threats.
- STRIDE is used for threat modeling to evaluate threats and their associated attacks. Although it was used for software evaluation in information systems, some work has allowed its adaptation to blockchain-based systems.
- DREAD is another methodology for threat modeling. It is based on evaluating the impact of the following threats.
- OSSTMM is a methodology used for security testing.
- Six Sigma is a strategy to improve organizational process that can be used to improve security processes in software development.

The frameworks that can be considered in the field of software security are the following:

- Zachman is a framework that allows the organization of controls according to an enterprise architecture.
- COBIT is a framework for the governance and management of technologies.
- COSO is a framework for internal control and enterprise risk management.
- SABSA is a methodology for developing an enterprise information security architecture and service management.
- ISO 27034 is a standard that deeply analyzes the security of applications developed in information technologies [20].
- NIST 800-64 is a guideline for security considerations in the system development life cycle [21].

Enhancing security sometimes stems from the organization's initiatives, while in other instances, it arises from the need for compliance from external sources. This compliance depends on the vertical in which the software is developed. For example, in the case of healthcare, an HIPAA compliance is required. Some compliance standards are as follows:

- SoX regulates the presentation of financial documentation.
- BASELII defines a reference framework for financial risk management.
- GLBA focuses on security compliance in the financial sectors.
- HIPAA defines a framework for health records.

- The DATA protection Act is the implementation of GDPR in the UK which focuses on controlling the use of personal information by organizations, businesses, or government.
- The Computer misuse Act focuses on the protection of personal data held by organizations against unauthorized access or modification.

Incorporating a security perspective into the software development process can diminish software vulnerabilities and mitigate the potential risk of security incidents. Proactive security strategies require continuous monitoring to assess and detect potential vulnerabilities that can be exploited by attackers. For example, Zhou et al. [22] propose a quantum neural network for software vulnerability detection. The proposal is based on the premise that the rapid growth of 5G networks increases the scale of unknown vulnerabilities. Therefore, the proposal is based on training a neural network based on code gadget labels. Several techniques and tools can be used to evaluate software security. A selection of these is presented as follows:

- Application Security Testing Orchestration (ASTO),
- Correlation tools,
- Test Coverage Analyzers,
- Mobile Application Security Testing (MAST),
- Interactive Application Security Testing (IAST),
- Static Application Security Testing (SAST),
- Dynamic Application Security Testing (DAST),
- Software Composition Analysis (SCA).

3.4. The Process of Cybersecurity Gathering Requirements on Software Development

Elicitation is the process of gathering understanding about the requirements of a software system. There are several models that can be used to elicit software requirements in the software development process. A selection of them is presented as follows:

1. Use case model: describes the behavior of a software system from the user's perspective, including the interactions between the user and the system.
2. Business process model: describes the processes and workflows of an organization. It can be used to elicit requirements by identifying the tasks and activities that the software system needs to support.
3. Data model: it is a graphical representation of the data that the software system manages. It can be used to elicit requirements by identifying the data entities, attributes, and relationships that are needed to operate it.
4. Object model: it is a graphical representation of the objects and classes that the software system uses. It can be used to elicit requirements by identifying the objects and their attributes and relationships.
5. Scenario model: describes the behavior of a software system in response to a particular scenario or use case. It can be used to elicit requirements by identifying the steps that the software needs to take to respond to a scenario.
6. State-transition model: describes the states and transitions of a software system. It can be used to elicit requirements by identifying the conditions under which the software system transitions from one state to another.
7. User interface model: describes the user interface of the software system. It can be used to elicit requirements by identifying the user interface elements and their interactions with the software system.

These techniques are more focused in the process of capturing functional and non-functional requirements. However, there are several standards, frameworks, and models that can be used for cybersecurity scenarios in the elicit software process:

1. Misuse case model is a variation of the use case model that focuses on the potential misuse of a software system. It can be used to identify and document potential abuse scenarios to ensure that the software system is designed to be secure [23].

2. Security Quality Requirements Engineering model (SQUARE) is a process-based model for eliciting, analyzing, and specifying security requirements in software systems. In the elicit phase, it is possible to identify potential abuse scenarios that the system might be vulnerable to [24].
3. Threat model is a framework for identifying and addressing security threats in software systems. For instance, STRIDE considers that the main attacks could be Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege. The framework can be used to develop abuse stories by identifying potential threats and vulnerabilities in the software system [25].
4. Process for Attack Simulation and Threat Analysis model (PASTA) is a risk-based approach to security testing and analysis. It can be used to develop abuse stories by identifying potential attack paths and scenarios that an attacker might use to exploit vulnerabilities in the software system.
5. NIST Cybersecurity Framework is a set of guidelines and best practices for improving cybersecurity in organizations. It can be used to develop abuse stories by identifying potential threats and vulnerabilities in the software system and developing strategies for mitigating those risks.
6. OWASP Top 10 is a list of the top 10 most critical web application security risks. It can be used to develop abuse stories by identifying potential vulnerabilities in web applications and software systems.

Threat modeling could help to define the possible threats that could affect the security of software products. Some tools to define threat modeling are STRIDE, SQUARE, and OCTAVE. There are several adaptations to threat models. For instance, Park et al. [26] proposes a threat modeling and valuation graph to provide a graphical representation related to the impact that an attack inflicts on an asset in terms of that impact, damage, recoverability, and likelihood. Gulati et al. [27], characterizes threats according to STRIDE to help non-technical better stakeholders relate these threats to their needs. Mead et al. (2005) [28] proposes the use of a methodology for quality requirements based on the SQUARE Process. This work highlights that define artifacts, business goals, risk assessment of impact, and the likelihood of threats affecting an organization's risk tolerance could be hard to follow for an Agile fast-based environment.

Misuse case is another technique to define negative interactions or behavior with the software functionalities. According to Whittle et al. (2008) [29], misuse cases are a way of modeling negative requirements. They can be used to model attacks on a system as well as the security mechanisms needed to avoid them. There are variants to the use of misuse cases. Yoo et al. [30] incorporates the goals of attackers, offering more realistic meanings to security threats, and mentions that any change in or addition of functional requirements requires the risk of being re-evaluated. Conducting abuse stories for threat modeling could help to define various abuse scenarios used for bad users or attackers [31,32]. For instance, in the development of E-Commerce Platforms, abuse stories could define ways in which the systems working to prevent fraudulent activities, such as phishing scams, fake product listings and conduct payment fraud.

4. Modelling Abuse Stories in Scrum Methodology

Based on the literature review, we can see that there is a set of methodologies and practices that have addressed the aspects related to secure software development [33]. An important aspect of the software development process is the need to cover the compliance of these methodologies, but it is also important to address aspects such as threat and risk modeling during the software development process to define the possible control to avoid cybersecurity attacks. Another important issue is the consideration of the human aspects in the software development process. Sometimes, the execution of code abstraction or code refactoring could add problems related to software security due to the replication of bad code or bugs within the code or vulnerabilities resulting from the misapplication of

abstraction concepts. Additionally, it is relevant to define a continuous monitoring process to validate security in software development.

In Figure 4, we propose a software security development process based on four stages: Governance, Modeling, Construction, and Monitoring. The four components cover the main aspects obtained for the literature review related to software security:

- (a) The use of software development in new environments;
- (b) The human factors in software development;
- (c) The management of cybersecurity in software development;
- (d) The process of cybersecurity gathering requirements in software development.

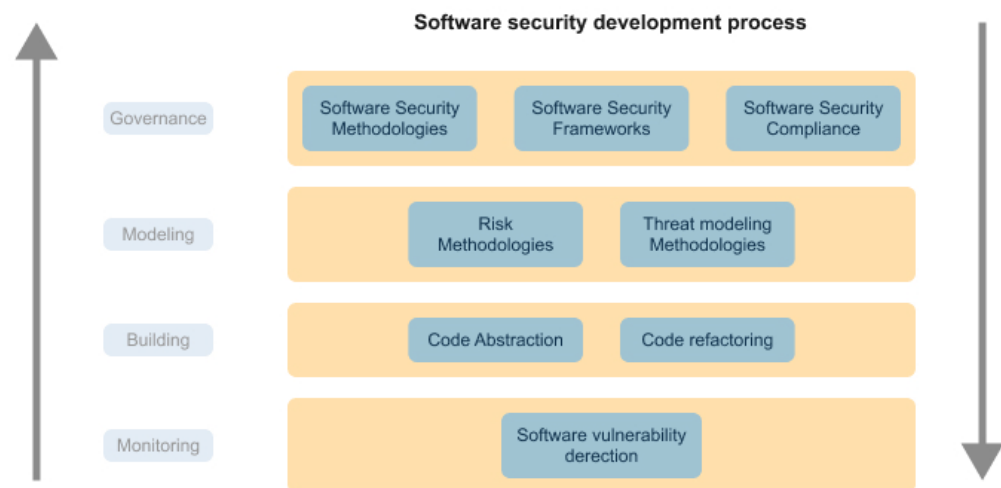


Figure 4. Proposal of Software Security Development Process (SSDP).

In the governance process, the objective is to define the most appropriate methodologies for the software development process. In the modeling phase, the definition of risk or threat modeling methodologies for the establishment of controls is defined. In the construction phase, the definition of good practices and control in the programming of the code is established to avoid the introduction of vulnerabilities at the code level. Finally, in the monitoring phase, a code security evaluation process is established through the selection of static or dynamic software analysis tools.

The process of eliciting software should inherently consider security factors as imperative requirements for any software system. Therefore, the process should include the four components mentioned in the proposal of the Software Security Development Process (SSDP). To tackle this concern, one effective approach involves integrating abuse stories into the software requirements gathering process. These abuse stories can be strategically developed between the requirements analysis and software design phases, both integral to the software development life cycle. It is crucial that abuse story cases undergo rigorous development involving threat assessment and risk analysis processes inherently embedded within the software design life cycle as well. Most articles accentuate the pronounced significance of abuse stories within the software elicitation process, underscoring their role in uncovering latent security vulnerabilities and associated risks tied to the software (Howard, 2002). A proposal about aspects that should be included in abuse stories in software development based on the four components of the Software Security Development Process are the following:

- (a) Governance (Elicitation Techniques for Abuse Stories): Techniques for eliciting abuse stories, including methods like interviews, brainstorming sessions, misuse cases and attack trees [34], should be tailored to align with organizational objectives and compliance.
- (b) Modeling (Mitigation of Abuse Scenarios): Abuse stories should primarily concentrate on establishing the mitigation of abuse scenarios through the implementation of

- security controls and other strategic measures. This proactive approach serves to pre-emptively thwart or minimize the potential impact of attacks on the software.
- (c) Building (Challenges in Eliciting Abuse Stories): Abuse stories should be oriented towards comprehending the motives of potential attackers, addressing a lack of domain knowledge, and bridging cross-cultural disparities [31]. Abuse stories should identify patterns, antipatterns, or bad practices utilized by software developers, which could be applied in a heuristic manner in the process of software development.
 - (d) Monitoring (Integration with Software Development Lifecycle): Integrating abuse stories into the software development lifecycle can aid in identifying and mitigating potential risks during the early stages of development [32]. Regular security assessments within each sprint can facilitate prompt rectifications in the code development or refactoring process.

Holistic Approach to Include Abuse Case Stories in Agile Methodologies

In the scrum development process, the product owner is responsible for creating and prioritizing user stories, including abuse stories. The abuse stories should be created based on potential security threats and vulnerabilities identified during the requirement elicitation process. The product owner should prioritize abuse stories based on their potential impact on the system. The scrum team should include the prioritized abuse stories in the sprint backlog along with other user stories. The development team should work with the product owner to ensure that the abuse stories are clearly defined and understood by the team. Abuse stories can be integrated into the scrum development process in the following way:

1. Implement abuse stories during the sprint: During the sprint, the development team should work on implementing the abuse stories along with other user stories. The team should ensure that they understand the security risks associated with the abuse stories and implement appropriate security controls to mitigate these risks (see Figure 5).

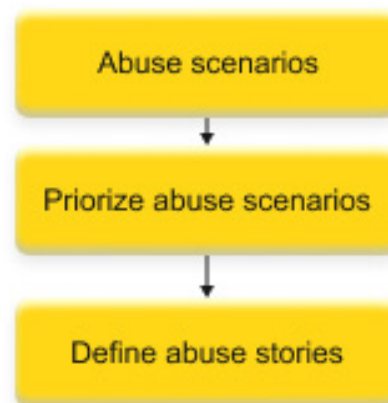


Figure 5. Steps to build abuse stories.

Sprint Planning is a component of the Scrum development framework, and abuse stories can be incorporated into the sprint planning process in Scrum. However, other Agile methodologies such as Extreme Programming (XP), Kanban, and Lean also have similar planning phases that could be used to integrate abuse stories. In XP, for example, there is a planning phase called “Iteration Planning” where the team selects a set of user stories to implement during the upcoming iteration. Abuse stories could be included in this planning phase and prioritized along with other user stories based on their impact on security. In Kanban and Lean, planning is a continuous process where work items are pulled into the development process as capacity becomes available. Abuse stories could be added to the Kanban board or included in the Lean value stream mapping process to ensure that security is considered throughout the development cycle. Overall, the key

is to ensure that abuse stories are included in the planning and prioritization process of any Agile development methodology being used. This ensures that security is integrated into the development process and considered at every stage of the software development lifecycle. Figure 6 shows an exercise of the use of abuse stories in the eliciting process using the event storming technique. The exercise was developed for students of software engineering in a course in software construction. A total of six subgroups were created. Each subgroup designed the process of abuse stories based on the development of the same application.

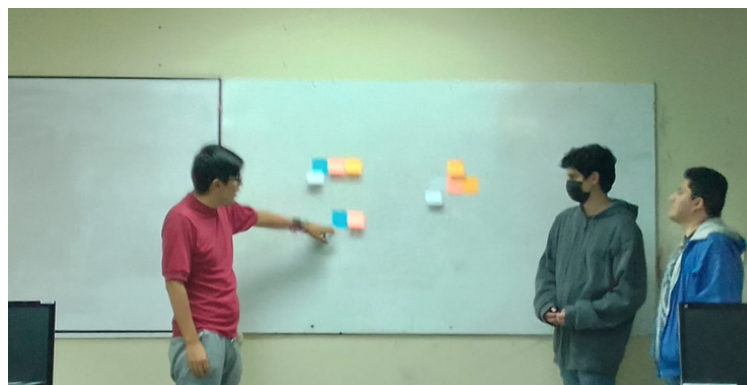


Figure 6. Development gathering of security requirements using abuse stories.

2. Conduct security testing: Once the abuse stories are implemented, the development team should conduct security testing to ensure that the system is secure. This can include techniques such as penetration testing, vulnerability scanning, or code reviews.

Abuse stories could be part of sprint planning meetings which are a key component of Agile development methodologies. Therefore, development teams can ensure that security considerations are built into each sprint. Also, considering threat modeling exercises in the Agile development process could be useful for identifying potential threats in a timely manner. Then, automated testing tools should focus on the abuse stories which are tested in the software development process. CI/CD pipelines automate the software development, testing, and deployment processes. After the integration of abuse stories into the CI/CD pipeline, security considerations can be built into each stage of the development process. A possibility is including Test-Driven Development (TDD) methodology to write test scenarios. One of the key principles of TDD is to write tests before writing the code. TDD could be used to test positive and negative scenarios. An important aspect of TDD methodology is that it is built under an Agile approach. Figure 7 shows an exercise of the use of abuse stories in the eliciting process using the event storming technique. The exercise was developed for students of software engineering in software quality assurance course. A total of seven subgroups were created. Each subgroup designed the process of evaluating the security requirements built in based on the same application proposed to the first group focus in the construction.

3. Review and adapt: After each sprint, the development team should review the abuse stories and the results of the security testing. Based on this feedback, the team should adapt their approach to security and make any necessary changes to the product backlog or development process. Finally, collaboration and communication in the Agile development process needs to be prioritized to ensure that abuse stories are identified, addressed, tested and corrected throughout the development process.

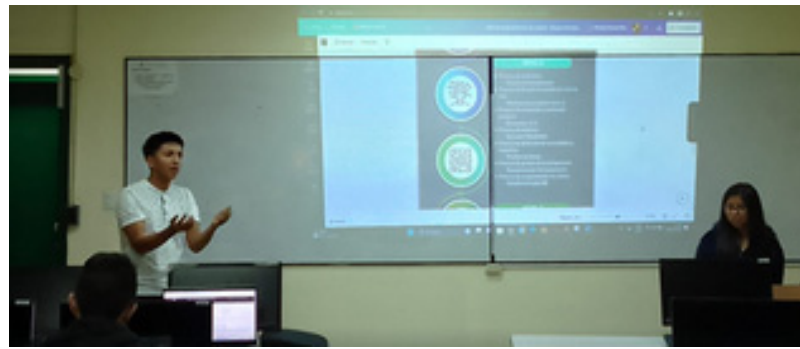


Figure 7. Evaluation of security requirements obtained from abuse stories.

5. Discussion

Solutions of software could be affected by factors such as short time to start its production, training in new languages of programming, and the need to protect sensitive data and the operation of IT solutions, which is more than just motivation of the technology department; in some cases, there are strict requirements for third-party regulations such as other companies or the government. The challenge in software security arises from different factors (see Figure 8) such as (i) the software development methodology used to build the application; (ii) cognitive bias, although security tests produced before the production step, human factors such as the experience of the developer or the possible cognitive biases by developers may induce software vulnerabilities in the final product [35]; (iii) each programming language has its own vulnerabilities that can affect the entire system [36], and therefore, it is important to consider the sensitivity and vulnerability to soft error when an application is developed; (iv) applications using functionalities or sharing data with third parties; consequently, one aspect to consider in software development are the vulnerabilities of third-party components [37]; (v) open-source versus licensed software, since open-source solutions contain libraries that allow reduction in production time by a pre-build of components, but that can also introduce vulnerabilities; therefore, it is important that developers pay attention to aspects such as the revision numbers of downloaded packages to avoid increasing the risk of security problems [38]; (vi) handling of sensitive data by applications. In several countries, regulations have been adopted for the protection of personal data and have generated the need for developers to consider security practices for data management [39].

These scenarios generate some challenges in software security:

- Timing to remediate code vulnerability;
- Identification of flaws or errors in code;
- Identification of good practices in the software development cycle;
- Protection of data sharing.

Several methodologies for software development have been proposed in recent years for organizations, and the development of software must consider in its initial stage certain levels of security. But this is sometimes not fulfilled due to different factors, including the lack of knowledge from developers regarding cybersecurity assuming that the team performing the tests can determine security problems in the developed code, failure to determine possible security issues on the part of the software project manager, or cognitive bias because generally, from the perspective of a software developer, a more positive vision is adopted based on how the software should work correctly, and not related with the flaws it could have. However, the incorporation of an approach to gathering information about security requirements for the software solutions inside the software development process is not new. In the elicit phase, it is common that software development teams include abuse stories and security requirements [40]. Abuse stories are a type of user stories that describe scenarios in which the software is intentionally misused or exploited. They help to identify

potential security vulnerabilities and other risks associated with software development projects.

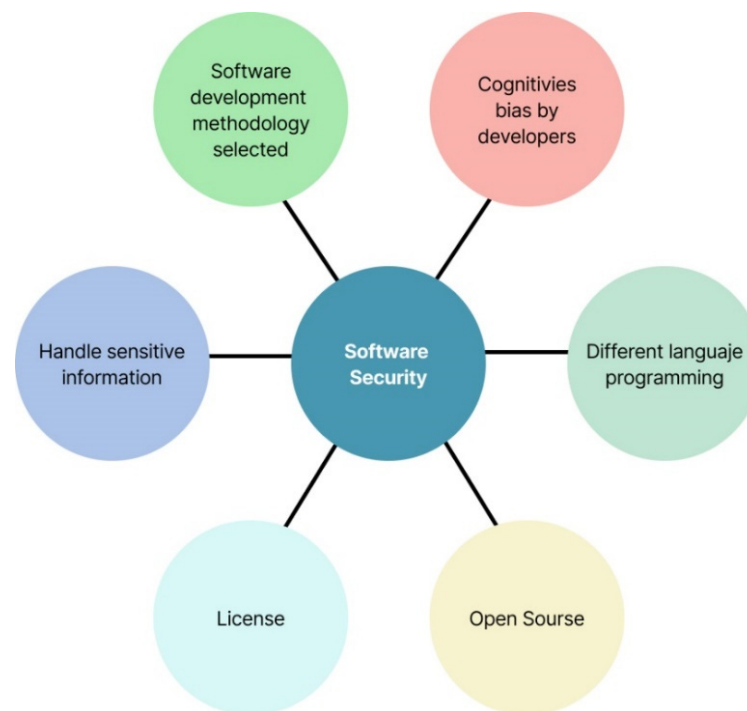


Figure 8. Critical factors in software development security.

Abuse stories can also help in the identification of the security requirements and aid software development teams in implementing adequate measures to mitigate the risks associated with potential security vulnerabilities. Some examples of abuse stories include attempting to access confidential data without authorization, modifying data without permission, and bypassing security controls. However, other techniques are available; for instance, attack trees that are more structured and involve breaking down a potential attack into a hierarchical tree-like structure. They are often used to help analyze and prioritize potential threats and can be useful for identifying specific steps that an attacker might take to exploit a vulnerability. One of the main advantages of abuse stories is that they are easy to understand and communicate to non-technical stakeholders, such as business leaders or end users.

The conducted experimentation process showed an enhancement in the acquisition of security requirements and controls that the application should encompass which were initially overlooked by the students during the software creation process. However, the main issue observed was that when the abuse stories process was included, the students required more time as they were unsure of its application. Subsequently, by introducing the use of color-coded postfixes and employing the event storming technique, the process was revisited, leading to improved acquisition of security aspects. Lastly, a third exercise was centered around utilizing UML to visually analyze abuse stories, resulting in a significant improvement in incorporating attack paths and necessary security mechanisms to mitigate abuse scenarios. The choice to use UML was based on the students' familiarity with it, given its utilization in programming courses. During the testing phase, the second group of students observed that the security aspects were appropriately addressed, leading to a satisfactory compliance level in terms of application security.

Abuse stories are just like conventional user stories, except they represent what an attacker would do in a software product. Therefore, an abuser story could have a similar syntax to that of the user story; it could help to create more consistent abuser stories for Agile software teams. Therefore, abuse stories should be included in product backlogs

working with the product owner, developers, and security experts. Abuse stories are distinct from misuse cases as they focus on malevolent activities, in contrast to potential misuse scenarios.

Limitation of Abuse Stories on Software Development

Abuse stories can fail in the elicit software process for several reasons:

1. **Lack of domain knowledge:** Eliciting abuse stories requires a good understanding of the domain in which the software system is being developed. If the requirement analyst does not have sufficient domain knowledge, they may miss important scenarios that could lead to security vulnerabilities.
2. **Incomplete or inaccurate scenarios:** If the abuse scenarios are incomplete or inaccurate, they may fail to identify all potential security vulnerabilities. This could be due to a lack of understanding of the software system, a failure to consider all possible misuse cases, or a failure to identify all possible attack vectors.
3. **Failure to consider the motivations of attackers:** When eliciting abuse stories, it is important to consider the motivations of potential attackers. If the abuse scenarios fail to consider the motivations of attackers, they may miss important scenarios that could lead to security vulnerabilities.
4. **Cultural differences:** If the software system is intended for use in different cultures, it is important to consider cultural differences when eliciting abuse stories. Failure to consider cultural differences could result in the failure to identify potential security vulnerabilities.
5. **Failure to integrate abuse stories into the software development lifecycle:** If abuse stories are not integrated into the software development lifecycle, they may be overlooked or not considered until later in the development process. This could result in the failure to identify potential security vulnerabilities early in the development process, which could be more costly to fix later.

6. Conclusions

From the point of view of security (specifically software security), one of the problems is that insecure software development is maintained for various reasons, including lack of experience of software developers in security by design, the TDD test focus on functionality and not on security in code, lack of use of methodology to develop software security, errors in code abstraction processes or cognitive biases in software development. Abuse stories represent a valuable technique within software development, aiding developers in recognizing malevolent users, attackers, and their potentially harmful interactions with the software.

Therefore, the problem of software vulnerabilities remains today in various applications and information systems, and the problem of cyber attacks continues to grow even when organizations are trying to improve network security or information security. Software security focuses on the idea of making software secure based on best practices or methodologies, but also encompasses deep analysis of code errors using static analysis (SATS), dynamic analysis (DAST), or mobile analysis (MAST). Additionality seeks to focus on programming errors by developers. Therefore, organizations need to consider software security as an organizational task and consider the software security development process, methodologies, modeling, software vulnerability detection, and the human factor for developers. TDD tests the behavior of software systems. This means that unit tests are written before coding. If some specific code is not present, the test fails; the code should be developed to run the test again. This could be useful to validate the compliance of security requirements in the software.

In the elicit software development process, abuse case stories and antipatterns in software could be used for identifying and addressing potential problems. Abuse case stories are focused on identifying potential abuse scenarios that could threaten the security

and integrity of software. By developing abuse stories, development teams can proactively identify potential security vulnerabilities and take steps to address them.

In some cases, abuse case stories may be used to identify potential antipatterns in software development. For instance, an abuse story can reveal a potential security vulnerability that is being addressed through a common but ineffective solution (antipattern). The relationship between abuse case stories and antipatterns in software highlights the importance of taking a proactive and holistic approach to software development. By identifying and addressing potential issues early in the development process, organizations can create more secure, efficient, and effective software that meets the needs of users and stakeholders. Antipatterns in software are common solutions to recurring problems that are often ineffective and can lead to negative consequences. Identifying and avoiding antipatterns in software development can avoid common pitfalls and create more effective and efficient software.

Abuse stories should be developed after user stories are created. Brainstorming is one approach that can be used to elicit abuse stories in the elicit software process, but it is not necessarily the best or only approach. There are several other techniques that can be used, depending on the context and the preferences of the stakeholders involved in the project. For example, interviews with stakeholders, a review of system requirements and related documents, and an analysis of historical incidents can also be used to identify potential abuse scenarios. Additionally, the use of threat modeling techniques, such as attack trees or misuse cases, can also help to identify potential vulnerabilities and abuse scenarios. In this context, Williams et al. [32] propose the development of abuse stories based on a list of keywords from threat modeling, attack patterns, and Common Weakness Enumeration. The study shows better results when creating abuse stories than using brainstorming.

Thus, it is important to use a variety of techniques to elicit abuse stories, as this can help to ensure that a wide range of potential scenarios are considered. The specific techniques used depend on the project's goals, the stakeholders involved, and the complexity of the software being developed. Ultimately, the goal is to identify as many potential abuse scenarios as possible in order to develop a more secure and resilient software system. But an important point is the fact that functional and security requirements should not be taken in the same session, because different stakeholders are needed in the elicit process.

Abuse stories implemented correctly within the development of the software process substantially mitigate errors during the phases of the software development cycle with the certainty that costs will be reduced because no corrections will be implemented in the code. The approach adopted by developers and testers must cover a multitude of scenarios from the perspective of not only the functionality of the software but also an attacker and their motivations for incurring developed software.

Author Contributions: Conceptualization, R.A. and J.T.; methodology, R.A.; formal analysis, R.A.; investigation, R.A.; writing—original draft preparation, J.T.; writing—review and editing, J.T. and J.M.; visualization, L.A.; supervision, I.O.-G.; project administration, I.O.-G.; funding acquisition, J.T. and R.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Escuela Politécnica Nacional from Ecuador, grant research project PIS-22-19. And the APC was funded by the same project and institution.

Data Availability Statement: Not applicable.

Acknowledgments: Students of the courses of software quality assurance and construction and evolution of software of the University of the Armed Forces-ESPE who collaborated with this study.

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

References

1. Security Magazine. Poor Software Costs the US 2.4 Trillion, Security Magazine RSS. 2022. Available online: <https://www.securitymagazine.com/articles/98685-poor-software-costs-the-us-24-trillion> (accessed on 10 August 2023).
2. Islam, S.; Falcarin, P. Measuring security requirements for software security. In Proceedings of the 2011 IEEE 10th International Conference on Cybernetic Intelligent Systems (CIS), London, UK, 1–2 September 2011; pp. 70–75. [CrossRef]
3. McGraw, G. Software security. *IEEE Secur. Priv.* **2004**, *2*, 80–83. [CrossRef]
4. Sametinger, J. Software Security. In Proceedings of the 2013 20th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS), Scottsdale, AZ, USA, 22–24 April 2013; p. 216. [CrossRef]
5. Byers, D.; Shahmehri, N. Design of a Process for Software Security. In Proceedings of the Second International Conference on Availability, Reliability and Security (ARES'07), Vienna, Austria, 10–13 April 2007; pp. 301–309. [CrossRef]
6. Fujdiak, R.; Mlynek, P.; Mrnustik, P.; Barabas, M.; Blazek, P.; Borcik, F.; Misurec, J. Managing the Secure Software Development. In Proceedings of the 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Canary Islands, Spain, 24–26 June 2019; pp. 1–4. [CrossRef]
7. Braz, L.; Bacchelli, A. Software security during modern code review: The developer's perspective. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022), Singapore, 14–18 November 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 810–821. [CrossRef]
8. Hope, P.; McGraw, G.; Anton, A.I. Misuse and abuse cases: Getting past the positive. *Secur. Priv.* **2004**, *2*, 90–92. [CrossRef]
9. Thomas, S. Abuser Story—User Stories to Prevent Hacking, It's a Delivery Thing. 2013. Available online: <https://itsadeliverything.com/abuser-story-user-stories-to-prevent-hacking> (accessed on 10 August 2023).
10. Sujee, S.; Solanki, R.; Dalwai, T. Technology Innovations for Business Growth—Impact of AI and Blockchain on Financial Services. In *Explore Business, Technology Opportunities and Challenges after the COVID-19 Pandemic*; ICBT 2022. Lecture Notes in Networks and Systems; Alareeni, B., Hamdan, A., Eds.; Springer: Cham, Switzerland, 2023; Volume 495. [CrossRef]
11. Bagchi, S. Security for Software on Tiny Devices. In *System Dependability and Analytics*. Springer Series in Reliability Engineering; Wang, L., Pattabiraman, K., Di Martino, C., Athreya, A., Bagchi, S., Eds.; Springer: Cham, Switzerland, 2023. [CrossRef]
12. Ren, Y.; Leng, Y.; Qi, J.; Sharma, P.K.; Wang, J.; Almahadmeh, Z.; Tolba, A. Multiple cloud storage mechanism based on blockchain in smart homes. *Future Gener. Comput. Syst.* **2021**, *115*, 304–313. [CrossRef]
13. Gopali, S.; Khan, Z.A.; Chhetri, B.; Karki, B.; Namin, A.S. Vulnerability Detection in Smart Contracts Using Deep Learning. In Proceedings of the 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), Los Alamitos, CA, USA, 27 June–1 July 2022; pp. 1249–1255. [CrossRef]
14. Horvath, R.; Nedbal, D.; Stieninger, M. A Literature Review on Challenges and Effects of Software Defined Networking. *Procedia Comput. Sci.* **2015**, *64*, 552–561. [CrossRef]
15. Aziz, N.A.; Mantoro, T.; Khairudin, M.A.; Murshid, A.F.B.A. Software Defined Networking (SDN) and its Security Issues. In Proceedings of the 2018 International Conference on Computing, Engineering, and Design (ICCED), Bangkok, Thailand, 6–8 September 2018; pp. 40–45. [CrossRef]
16. Wu, Y. Application of Data Encryption Technology in Computer Software Testing. In *The 2021 International Conference on Smart Technologies and Systems for Internet of Things; STSIoT 2021*. Lecture Notes on Data Engineering and Communications Technologies; Ahmad, I., Ye, J., Liu, W., Eds.; Springer: Singapore, 2023; Volume 122. [CrossRef]
17. Tøndel, I.A.; Cruzes, D.S. Continuous software security through security prioritisation meetings. *J. Syst. Softw.* **2022**, *194*, 111477. [CrossRef]
18. Brasoveanu, R.; Karabulut, Y.; Pashchenko, I. Security Maturity Self-Assessment Framework for Software Development Lifecycle. In Proceedings of the 17th International Conference on Availability, Reliability and Security (ARES'22), Vienna, Austria, 23–26 August 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 1–8. [CrossRef]
19. Ardagna, C.A.; Bena, N.; de Pozuelo, R.M. Bridging the Gap between Certification and Software Development. In Proceedings of the 17th International Conference on Availability, Reliability and Security (ARES'22), Vienna, Austria, 23–26 August 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 1–10. [CrossRef]
20. Lipner, S.B. Security assurance. *Commun. ACM* **2015**, *11*, 24–26. [CrossRef]
21. Chen, S.J.; Pan, Y.C.; Ma, Y.W.; Chiang, C.M. The Impact of the Practical Security Test during the Software Development Lifecycle. In Proceedings of the 2022 24th International Conference on Advanced Communication Technology (ICACT), PyeongChang Kwangwoon_Do, Republic of Korea, 13–16 February 2022; pp. 313–316. [CrossRef]
22. Zhou, X.; Pang, J.; Yue, F.; Liu, F.; Guo, J.; Liu, W.; Song, Z.; Shu, G.; Xia, B.; Shan, Z. A new method of software vulnerability detection based on a quantum neural network. *Sci. Rep.* **2022**, *12*, 8053. [CrossRef]
23. Damodaran, M. Secure Software Development Using Use Cases and Misuse Cases. *Issues Inf. Syst.* **2006**, *7*, 150–154.
24. Mead, N.R.; Viswanathan, V.; Padmanabhan, D.; Raveendran, A. Incorporating Security Quality Requirements Engineering (SQUARE) into Standard Life-Cycle Models, SEI Technical Note CMU/SEI-2008-TN-006. 2008. Available online: <http://www.sei.cmu.edu> (accessed on 10 August 2023).
25. Yuan, X.; Borkor, E.N.; Yu, H. Developing abuse cases based on threat modelling and attack patterns. *J. Softw.* **2015**, *10*, 491–498. [CrossRef]
26. Park, K.Y.; Yoo, S.G.; Kim, J. Security Requirements Prioritization Based on Threat Modeling and Valuation Graph. *Commun. Comput. Inf. Sci.* **2011**, 142–152.

27. Gulati, A. Proposing Security Requirement Prioritization Framework. *Int. J. Comput. Sci. Eng. Appl.* **2012**, *2*, 27–37. [[CrossRef](#)]
28. Mead, N.R.; Stehney, T. Security Quality Requirements Engineering (SQUARE) Methodology. *ACM SIGSOFT Softw. Eng. Notes* **2005**, *30*, 1–7. [[CrossRef](#)]
29. Whittle, J.; Wijesekera, D.; Hartong, M. Executable misuse cases for modeling security concerns. In Proceedings of the 30th International Conference on Software Engineering (ICSE'08), Leipzig, Germany, 10–18 May 2008; Association for Computing Machinery: New York, NY, USA, 2008; pp. 121–130. [[CrossRef](#)]
30. Yoo, S.G.; Vaca, H.P.; Kim, J. Enhanced Misuse Cases for Prioritization of Security Requirements. In Proceedings of the 9th International Conference on Information Management and Engineering, Barcelona, Spain, 9–11 October 2017.
31. Wei, C. *Misuse Cases and Abuse Cases in Eliciting Security Requirements*; University of Auckland: Auckland, New Zealand, 2005.
32. Imano Williams, X.Y.; McDonald, J.T.; Anwar, M. A Method for Developing Abuse Cases and Its Evaluation. *J. Softw.* **2016**, *11*, 520–527. [[CrossRef](#)]
33. S-SDLC. Introducing Secure Software development Life Cycle (S-SDLC). 2022. Infosec Institute. Available online: <http://resources.infosecinstitute.com/intro-secure-software-development-life-cycle> (accessed on 10 August 2023).
34. Hadavi, M.A.; Hamishagi, V.S.; Sangchi, H.M. Security Requirements Engineering; State of the Art and Research Challenges. *Lect. Notes Eng. Comput. Sci.* **2008**, *1*, 2168.
35. Panek, C. Understanding security layers. In *Security Fundamentals*; Wiley: Hoboken, NJ, USA, 2020; pp. 1–31. [[CrossRef](#)]
36. Cerveira, F.; Fonseca, A.; Barbosa, R.; Madeira, H. Evaluating the Inherent Sensitivity of Programming Languages to Soft Errors. In Proceedings of the 2018 14th European Dependable Computing Conference (EDCC), Iasi, Romania, 10–14 September 2018; pp. 65–72. [[CrossRef](#)]
37. Mahrous, H.; Malhotra, B. Managing Publicly Known Security Vulnerabilities in Software Systems. In Proceedings of the 2018 16th Annual Conference on Privacy, Security and Trust (PST), Belfast, Ireland, 28–30 August 2018; pp. 1–10. [[CrossRef](#)]
38. Wen, S.-F. Software security in open source development: A systematic literature review. In Proceedings of the 2017 21st Conference of Open Innovations Association (FRUCT), Helsinki, Finland, 6–10 November 2017; pp. 364–373. [[CrossRef](#)]
39. Li, S.-C.; Chen, Y.-W.; Huang, Y. Examining Compliance with Personal Data Protection Regulations in Interorganizational Data Analysis. *Sustainability* **2021**, *13*, 11459. [[CrossRef](#)]
40. Howard, M.; LeBlanc, D.C. *Writing Secure Code*, 2nd ed.; Microsoft: Redmond, WA, USA, 2002.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.