International Journal of Applied Engineering Research

Agile Security Requirements Engineering for Conservation A Proposed Framework

Muhammad Ali Abid[1], Mudasir Mahmood[2*], Muhammad Ijaz Khan[2],
Saadat Ullah[2], Muhammad Sajjad Hussain[2], Asad Ullah[2], Muhammad Mustafa[2]
ali.abid@uad.edu.pk, mudasir@gu.edu.pk, ijazkhan@gu.edu.pk, ksaadat125@gmail.com,
muhammadsajjadhussain@gmail.com, asadullahpushia@gmail.com, attleramustafa@gmail.com
[1]*Department of Internet of Things, University of Agriculture, Dera Ismail Khan.*
[2]*Institute of Computing and Information Technology, Gomal University, Dera Ismail Khan.*
Pakistan 29050
[2*]Corresponding Author: mudasir@gu.edu.pk

Abstract: ***This research paper proposes a novel method for extending Agile principles to facilitate the integration of security requirements engineering into Agile Development while maintaining its iterative and responsive nature. The proposed method leverages Agile practices such as the creative planning game and coding rules to create two new types of Agile User stories: heckler stories to represent threat repercussions and security-related user stories to represent security functionality. These additions aid in communicating and implementing explicit coding and design standards for software development projects that include security requirements, as well as accommodating special security-related user stories. The proposed method was evaluated in a university student project to test its effectiveness.***

Keywords: ***Security Engineering Requirements, Hackler Stories, Software Security Management and its protection, Security related User Stories, Agile Software Development methodology (ASDM), Software Engineering Management (SEM)***

1. Introduction

For several decades, software project management has been guided by the technical coherence method, which employs a sequential approach known as the waterfall lifecycle. However, according to the latest CHAOS report [26], a significant percentage of software projects still fail, with less than 69% achieving success. One of the reasons for this is the limited opportunity for customer input and testing, as these activities are typically performed at the end of the project. This can result in issues related to misunderstood requirements and poor user comprehension of the software flow, which are also common problems with agile, template-based, and document-driven software development.

Traditional software development approaches often require a significant amount of paperwork, which can be time-consuming and resource-intensive [10]. Moreover, prevailing security engineering standards and criteria assume a stable development environment with clearly

defined, fixed, and documented software project strategies and security requirements. These standards are designed around a progressive, non-iterative lifecycle, which can be challenging for agile methodologies that prioritize iterative development and responsiveness.

To address these issues, there is a need for a more flexible and adaptable approach to software project management that integrates security engineering practices while maintaining the iterative and responsive nature of agile methodologies. This research paper proposes a new framework for security requirements engineering in agile software development that leverages the strengths of both approaches. The proposed framework combines agile principles such as user stories and iterative development with security engineering best practices, including threat modeling and risk assessment, to enable the development of secure software that meets customer requirements and expectations. The effectiveness of the proposed framework was evaluated through a case study, which demonstrated its potential for improving software project success rates and enhancing software security.

The ISO 15408 Common Criteria (CC) were originally developed for military purposes, resulting in significant documentation requirements. However, CC has been criticized for its resource-intensive and time-consuming nature. Consequently, attempts have been made to design a more flexible CC process. This is a common misconception of the Common Criteria for Information Technology Security Evaluation (2012) that allows independent security evaluation results to be compared. It sets forth a uniform set of requirements for IT product security capabilities and assurance measures. The CC can be used to aid in the creation, evaluation, and/or procurement of IT products with security features. However, users are cautioned not to abuse the standard's flexibility. Although the CC is intended to be adaptable, challenges still exist.

Given the low success rate of software projects, a new category of project management methodologies known as "agile methods" has emerged. These methods are iterative and leverage software engineering's "soft" nature. The new methodologies are based on the principles of "speculate, collaborate, and learn," rather than "plan, design, and build."

It is a practical approach to add more effective phases and document items to the agile development, iterative, and rapid feedback development processes. There are no provisions for software engineering or security requirements, for example. In a recent study, researchers examined how systems security engineering approaches align or do not align with agile methodologies.

One option is to combine Agile Software Development with traditional system security engineering, but this would limit the benefits of agile software development. We propose a balanced approach between documentation-centric and plan-driven, traditional software security engineering approaches. Our approach is based on the findings of a previous

investigation into how Agile approaches the activities and requirements of system security, as defined by SSE, Capability Maturity Model (CMM), and Common Criteria (CC).

Our technique is discussed in detail. By introducing new phases, it covers the agile planning game's approach to identifying and prioritizing occupational requirements. Two new concepts for understanding agile user stories' security are "addictive user stories" and "security-related user stories."

The objective of this research paper is to explicate the importance of addressing security-related user stories during software development, particularly to developers and consumers. The paper showcases the efficacy of our approach by presenting a case study wherein safety engineering students utilized our method to formulate requirements for a secure communication system they were developing [17]. The subsequent section delves into the agile planning process and elucidates on the potential extensions that can be incorporated into the process. The paper culminates with conclusions drawn from the study and outlines avenues for future research, with Sections 4 and 5 presenting analogous findings.

2. Agile Planning Game: An Overview

The Agile planning game is a planning process that comprises two distinct planning approaches. Firstly, the statement planning method is employed to establish a structural declaration plan, which involves identifying system requirements, specifying project scope, and outlining performance criteria. Secondly, the speedy iteration plan is derived from the release plan and is used to add or remove new requirements and enhance existing ones [32].

To create the structural declaration plan, the statement planning approach necessitates the use of user stories. These stories are brief, quantifiable, and testable descriptions of the users' needs, typically written on index cards and employed in Agile projects. The 3Cs User Stories methodology, which entails using Cards, Conversations, and Confirmation, is utilized to obtain most of the consumer information by selecting it from a list [4, 28].

3. Extending the Planning Game

In this research paper, we propose the integration of security engineering activities into the planning game to enhance the security of Agile software development. However, we caution against transforming Agile into a heavy, document-centric, plan-driven methodology as it would diminish its effectiveness. Thus, a delicate balance between security and Agile priorities must be struck, and the appropriate extensions must be tailored to the specific project's needs and

goals. Our experience with security engineering has led us to believe that these goals are critical [23, 31].

To achieve this integration, we recommend the following guidelines. Firstly, new activities should align with Agile methodologies and terminology as much as possible. Secondly, the activities should promote iterative work, emphasizing simplicity in rewriting results rather than relying solely on documentation [23, 31]. Thirdly, the output of the requirements process should be easy to follow during the testing and coding phases of the iteration. Fourthly, relevant sections of the product should be modified requirements to facilitate external evaluation [21]. Fifthly, gathering activities should encourage the development of proactive security measures. Finally, risk analysis should be considered when establishing requirements to identify potential security threats [30].

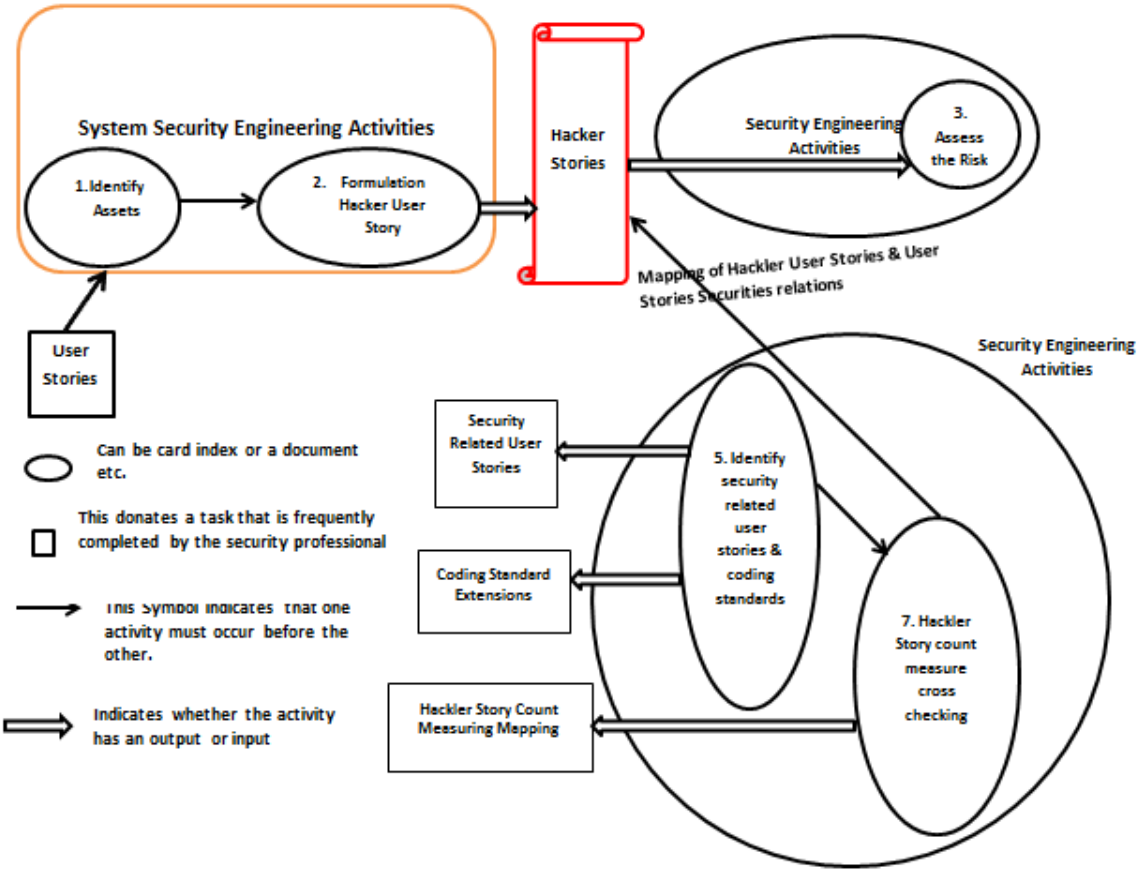


Figure 1. Out puts of expanded Agile Planning Game

To achieve our objectives, we propose a diagram outlining the recommended steps for including security requirements in the Agile Planning Game:

1. Identify assets with high security risks.
2. Simulate heckler stories using the Threat Scenarios mode.
3. Conduct a risk assessment for each heckler story.
4. Negotiate the heckler user stories.
5. Define security-related user stories.
6. Define coding standards associated with security.
7. Cross-check countermeasures for each heckler story.

Figure 1 depicts the recommended steps for including security requirements in the Agile Planning Game, which are discussed in detail below. To illustrate the application of this approach, we also present artifacts generated during its implementation in a secure negotiation system.

1 - Assets with high security risks must always be identified.

In Agile software development, the primary objective of a team is to identify and validate critical assets within the system under iterative development. Assets refer to anything that holds value for the organization or system users and are listed on the planning whiteboard. In the context of a negotiation system, an example of such an asset would be "confidential negotiation approaches" that exist within the system's architecture [17].

2. Simulation of heckler stories (Threat scenarios)

Heckler stories are security-related narratives that communicate potential risks to critical assets in a format and language that is easily understandable by both agile developers and customers. Similar to user stories, they are represented on index cards using the 3Cs approach, which facilitates communication between developers and customers. The creation of threat scenarios requires a thorough understanding of security risks and attacks, and hence, the involvement of a security engineer is crucial. A Heckler story outlines a hypothetical adverse interaction between a threat actor and a system that, if successful, would put the system's owner or user at risk. The customer team discusses the Heckler stories with the Agile Team to ensure their relevance and significance. Tools such as assault patterns can be used to develop these stories [11].

We argue that for agile projects, security requirements should be derived from detailed Heckler stories rather than general ones. Our reasoning is based on two factors. Firstly, it is impossible to test the mitigation of risks based on general types of threats. Developers must be able to demonstrate mitigation through test results to convince customers that a particular threat category will be objectively addressed [27]. Despite the differences in the nature of threats represented by Heckler stories and the Planning Game, a system architecture based on both narratives can be generic enough to address both types of risks. Heckler stories provide a common and measurable basis for developers to integrate ongoing security requirements into the agile planning game.

3. Risk assessment for the Heckler narrative

The client and security teams evaluate the risks identified in the Hackler stories and allocate a risk level. The probability and effects of the threat being realized for each Hackler story are calculated. The risk of the product is determined by multiplying the threat probability with its consequence. In evaluating the business impact of the risks, the expertise of the customer team is crucial. This is in line with sources [19,34].

The attractiveness of Hackler stories is discovered through action and affected encounters, which is typically not fully developed. This is achieved through economy of setting, brief storytelling, and the avoidance of a convoluted plot. Despite their limited scope, Hackler stories are widely assessed for their ability to present their characters and themes in a complete or fulfilling manner.

The Hackler stories are categorized into four quadrants based on the likelihood and consequences of the events. After the assessment, yellow or red stickers are placed on the Hackler Stories index to indicate the level of risk on the maps. These findings are consistent with source [18].

4. Negotiation between the Heckler User stories

The process of iteration planning commences by allowing the customer to identify the hackler user stories that necessitate addressing in the forthcoming iteration after conducting a comprehensive risk assessment. This approach mirrors the typical agile planning game methodology utilized for selecting user stories for implementation. Subsequently, the agile developers and security engineers collaborate to estimate the duration required to mitigate the identified hacker stories provided. This estimation process is essential in determining the feasibility and viability of addressing the identified security risks within the iteration timeframe [15].

5. Defining user stories about security.

In this research paper, it is emphasized that Hackler stories are essential for functional security requirements in response to hacker stories. Unlike security-related user stories, Hackler stories require validation through module, system, and integration testing. The research stresses that security-related user stories mandate encryption for all interactions and transmission of papers during debates. Collaboration and communication are integral to the creation of user stories, and security stakeholders should be involved in this process. The research suggests that security needs can be tied to the story map to avoid scope creep. Acceptance criteria complement the user story and are necessary to ensure the story's success.

The research proposes that security patterns can assist in identifying the necessary security features to combat hacker stories. Developers can specify security standards through standard user stories in the technical stage. This defines the security function that needs to be developed for a given user story.

A significant difference between user stories and requirements is highlighted in the research. A typical requirement focuses on the product's functionality, whereas a security requirement is a

statement of the required security functionality that assures one of the software's many security attributes is met.

Furthermore, the research identifies security-related user stories as those that a third-party security researcher would be interested in. It is suggested that these user stories should be emphasized to distinguish them easily from other user stories. Finally, the research points out that no user stories on the list provided are linked to security outside of the ones identified as security-related.

6. Security Linked Coding Standards Definition

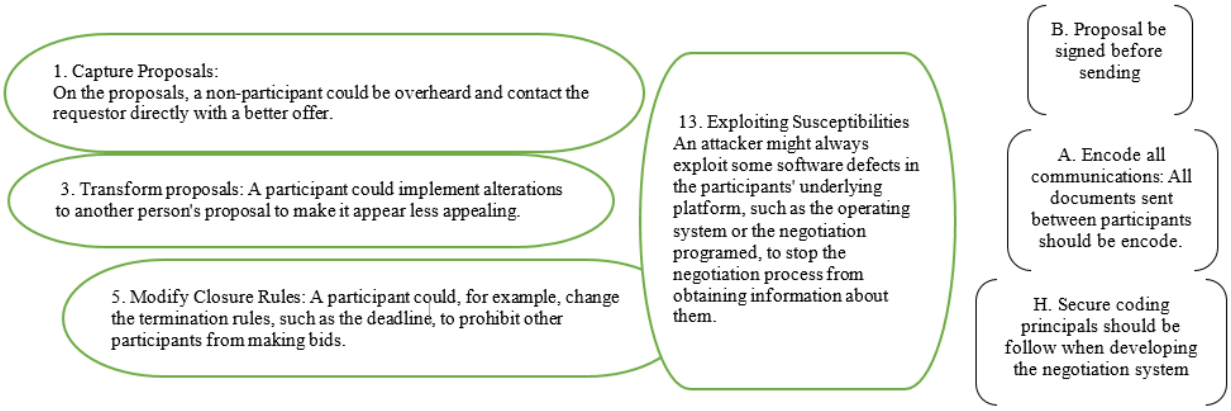
In the context of cybersecurity, it is not valid to dismiss a hacker's account of a security breach solely based on a customer's report. System-wide approaches are required to address major security threats, such as binary code injection through buffer overflow. Secure coding is a critical component of the software development lifecycle, and adherence to secure coding standards is necessary to prevent security flaws in software [2]. Such standards provide principles and guidelines to avoid, identify, and eliminate vulnerabilities that could compromise software security. Prior to the addition of new code, existing code must comply with the latest design or coding standards. For instance, an extension to the coding standard could dictate that "none of the dangerous C features are to be used in the code at any point."

7. Cross checking countermeasures in the Heckler story

To develop a secure application, it is essential to identify the attacks that the application may face in its business and technical environment [3, 12]. This practical approach enables the application to be designed with adequate countermeasures, such as security protection, user history protection, and verification activity protection. Adherence to coding standards is crucial to achieving software security. An example of a coding standard extension is the prohibition of using harmful C functions in the code. To combat major security threats, a system-wide approach is necessary.

Application of the technique to a student thesis project.

The present study involves the use of a risk assessment matrix to evaluate Hackler stories and molester stories based on their respective dangers and consequences. The primary objective of this study is to develop a secure agreement planner that can be utilized for negotiating Service Level Agreements (SLAs) in virtual organizations through the internet. The study's participants documented and analyzed the process's various artifacts using a whiteboard, including user stories, assets, and risk assessment findings. The risk assessment matrix was then populated with numerical values assigned to the Hackler stories, representing the initial validation of our SLA negotiation technique [20]. The Hackler stories countermeasure crosschecking table is situated at the bottom left of the whiteboard. Furthermore, Figure 2 depicts a few Hacklers and security-related user stories that were created during the study. The CORAS approach was selected by the students to conduct the risk analysis work, given their familiarity with it. This



study serves as a prospective example of the Common Criteria for Risk Analysis process improvement.

Figure 2: For a secure intercessor system, safety hacker stories are required.

3.2 Agile Process impact on other activities (Effects)

In the implementation phase of an agile iteration, developers are responsible for creating tests and implementing the system's stated user stories. Once the system is delivered to the customer, it is essential that they are informed of the assumptions made during the development process regarding potential threats and their sources, as well as the system's background. In addition to unit and acceptance tests, automated resistance testing and motionless analysis of the system's source code can be valuable.

Agile methodologies, which employ iterative development and prototyping, are frequently used in various industry projects as a lightweight development process that can adapt to changing requirements. Traditional software development approaches are insufficient in coping with

rapidly changing needs. Some critics argue that Agile disregards architectural and design considerations, leading to minor design decisions.

Operational deliverables from the planning game processor, such as hackler stories, secure design and coding standards, and security-related user stories, are used to identify and document these assumptions. These internal deliverables from the Planning Game's processor are adequate for determining and documenting such assumptions. However, Hackler's methodology does not provide any specific methods for creating these assumptions, although this could be a topic for future research. It is not reasonable to expect clients to have enough knowledge about security to document their assumptions. Hackler contends that such assumptions are ineffective in agile story-centric planning, testing, and implementation procedures.

3.3 Security Requirements and Roles Game of Engineering and Planning

To define the suggested process adaptation, it is essential to identify the roles of different participants involved in an agile project, where all team members work closely together [5]. The client team comprises domain experts, product managers, and end-users, while the customer team consists of the development team and safety engineer(s).

The first and most crucial step in building a robust security plan is understanding information security requirements. However, compliance requirements should not dictate the commitments that must be examined. Instead, the security commitments should align with the company's and customers' needs, which may be greater than what was initially set out [25]. These commitments can be categorized into Business Requirements, Regulatory Obligations, and Customer Omissions.

The client team is responsible for gathering requirements and prioritizing projects, while the development team consists of programmers, testers, and system analysts who help refine requirements and conduct acceptance tests. A security engineer, who may be one or more individuals, provides security knowledge to both the customer and the developers throughout the project.

A security requirement is a statement of necessary security functionality that ensures one or more of the software's security attributes are met. Security requirements are developed using industry standards, current regulations, and historical vulnerabilities [22]. Security requirements engineering is the primary concern of early-stage security engineering approaches. Implementing information security requirements helps companies be better prepared for security threats faced by both the company and its customers.

The primary responsibility of the security engineer is to assist the customer in identifying security requirements during the requirements phase. During implementation, the security engineer supports developers through training and pair programming, playing the role of devil's advocate by identifying potential hazards that must be considered in the system's production environment [24, 29].

4. Conclusions and Future Plans

Incompatibilities exist between the requirements for security engineering and the agile process approach. The latter involves a limited number of work outputs that are distributed iteratively, with security actions being integrated relatively normally. However, security engineers can still gain insights from the iterative development process.

Agile development, being iterative, enables the prompt assessment of the effectiveness of security requirement techniques and their execution through a sequence of user stories and design standards. Agile methodologies employ simple documentation tools such as index cards and whiteboard drawings.

The conflict between agile methodologies and security concerns can affect the quality attributes of complex or operation critical systems, impacting safety, high availability, and high performance. Basic agile methodologies may need to be adapted, and architectural and design standards must be prioritized.

Further research is necessary to comprehend how agile assurance approaches such as pair programming and testing can be employed to monitor security requirements. The Software Approach Improvement Network will initiate a preliminary validation via workshops (SPIN-Stockholm). As revealed in the first student project, adding more assistance to consider the system environment could enhance the process, although this should be viewed as a standard component of the process.

This paper elucidates how safety requirements can be preserved while maintaining agile planning, thereby completing projects on time instead of abandoning them.

References

1. De Lange, M. (2015). The playful city: using play and games to foster citizen participation.
2. Danchev, D. (2011). Malicious PDF files becoming the attack vector of choice.
3. Paige, R. F., Galloway, A., Charalambous, R., Ge, X., & Brooke, P. J. (2011). High-integrity agile processes for the development of safety critical software. *International Journal of Critical Computer-Based Systems*, *2*(2), 181-216.
4. Highsmith, J. (2013). *Adaptive software development: a collaborative approach to managing complex systems*. Addison-Wesley.
5. Demissie, S., Keenan, F., & McCaffery, F. (2016, June). Investigating the suitability of using agile for medical embedded software development. In *International Conference on Software Process Improvement and Capability Determination* (pp. 409-416). Springer, Cham.
6. Baca, D., & Carlsson, B. (2011, May). Agile development with security engineering activities. In *Proceedings of the 2011 International Conference on Software and Systems Process* (pp. 149-158).
7. NAZIR, R. (2021). Studying Software Architecture Design Challenges, Best Practices and Main Decisions for Machine Learning Systems.
8. J&rjens, J. (2009). Security and dependability engineering. In *Security and dependability for Ambient Intelligence* (pp. 21-36). Springer, Boston, MA.
9. Woody, C. (2013). Agile security-review of current research and pilot usage. *SEI White Paper*.
10. Gupta, A. D., Jaiswal, B. S., & Tewari, C. A. (2013). Security Requirements Engineering: Analysis and Prioritization. In *Proceedings of the International Conference on Software Engineering*

Research and Practice (SERP) (p. 1). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

11. Flechais, I., Mascolo, C., & Sasse, M. A. (2007). Integrating security and usability into the requirements and design process. *International Journal of Electronic Security and Digital Forensics*, *1*(1), 12-26.
12. Kotilainen, W. (2018). System-Level Checkpointing of Verification Tools.
13. Highsmith, J. (2013). *Adaptive software development: a collaborative approach to managing complex systems*. Addison-Wesley.
14. Setapa, S., Isa, M. A. M., Abdullah, N., & Ab Manan, J. L. (2010, December). Trusted computing based microkernel. In *2010 International Conference on Computer Applications and Industrial Electronics* (pp. 1-4). IEEE.
15. van Oorschot, Paul C. "Software security and systematizing knowledge." *IEEE Security & Privacy* 17.03 (2019): 4-6.
16. Elder, Sarah E., et al. "Structuring a comprehensive software security course around the OWASP application security verification standard." *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 2021.
17. Anwar Mohammad, Malik Nadeem, Mohammed Nazir, and Khurram Mustafa. "A systematic review and analytical evaluation of security requirements engineering approaches." Arabian Journal for Science and Engineering 44.11 (2019): 8963-8987.
18. Zarour, Mohammad, Mamdouh Alenezi, and Khalid Alsarayrah. "Software security specifications and design: How software engineers and practitioners are mixing things up." *Proceedings of the Evaluation and Assessment in Software Engineering*. 2020. 451-456.
19. Mishra, Aditya Dev, and Khurram Mustafa. "A Survey on Formal Specification of Security Requirements." *2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*. IEEE, 2021.
20. Xu, Yilin, et al. "A co-occurrence recommendation model of software security requirement." *2019 International Symposium on Theoretical Aspects of Software Engineering (TASE)*. IEEE, 2019.
21. Arogundade, Oluwasefunmi 'Tale, et al. "A study of existing use case extensions and experience: a systematic review." *International Journal of Computer Mathematics: Computer Systems Theory* 5.4 (2020): 263-281.
22. Viega, John. "20 years of software security." *Computer* 53.11 (2020): 75-78.
23. Koç, Güler, Murat Aydos, and Mehmet Tekerek. "Evaluation of trustworthy scrum employment for agile software development based on the views of software developers." *2019 4th International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2019.
24. Newton, Nathan, Craig Anslow, and Andreas Drechsler. "Information Security in Agile Software Development Projects: a Critical Success factor Perspective." *ECIS*. 2019.

25. Ionita, Dan, et al. "Towards risk-driven security requirements management in agile software development." *International Conference on Advanced Information Systems Engineering*. Springer, Cham, 2019.
26. NYMAN, NICK. "Threat Awareness in Agile Environments: Creating a Developer-Driven Threat Modeling Process for Agile Software Development Teams." (2020).
27. Bishop, Dave, and Pam Rowland. "Agile and secure software development: an unfinished story." (2019).
28. Ruiz, Jose Fran, et al. "Emergency systems modelling using a security engineering process." *Proc. of 3rd Int. Conf. SIMULTECH. SciTePress*. 2013.
29. ben Othmane, Lotfi, et al. "Extending the agile development process to develop acceptably secure software." *IEEE Transactions on dependable and secure computing* 11.6 (2014): 497-509.
30. Baca, Dejan, et al. "A novel security-enhanced agile software development process applied in an industrial setting." *2015 10th International Conference on Availability, Reliability and Security*. IEEE, 2015.
31. Zech, Philipp. "Risk-based security testing in cloud computing environments." *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. IEEE, 2011.
32. Ray, Mitrabinda, and Durga Prasad Mohapatra. "Risk analysis: a guiding force in the improvement of testing." *IET Software* 7.1 (2013): 29-46.
33. Mohammed, Nabil M., et al. "Exploring software security approaches in software development lifecycle: A systematic mapping study." *Computer Standards & Interfaces* 50 (2017): 107-115.
34. Singhal, Archana. "Integration analysis of security activities from the perspective of agility." *2012 Agile India*. IEEE, 2012.
35. Gurusamy, Kavitha, Narayanan Srinivasaraghavan, and Sisira Adikari. "An integrated framework for design thinking and agile methods for digital transformation." *International Conference of Design, User Experience, and Usability*. Springer, Cham, 2016.

