applied sciences

MDPI

Article

Identifying Key Activities, Artifacts and Roles in Agile Engineering of Secure Software with Hierarchical Clustering

Anže Mihelič [1,2,*], Tomaž Hovelja [2] and Simon Vrhovec [1]

[1] Faculty of Criminal Justice and Security, University of Maribor, 1000 Ljubljana, Slovenia
[2] Faculty of Computer and Information Science, University of Ljubljana, 1000 Ljubljana, Slovenia
* Correspondence: anze.mihelic@um.si

Abstract: Different activities, artifacts, and roles can be found in the literature on the agile engineering of secure software (AESS). The purpose of this paper is to consolidate them and thus identify key activities, artifacts, and roles that can be employed in AESS. To gain initial sets of activities, artifacts, and roles, the literature was first extensively reviewed. Activities, artifacts, and roles were then cross-evaluated with similarity matrices. Finally, similarity matrices were converted into distance matrices, enabling the use of Ward's hierarchical clustering method for consolidating activities, artifacts, and roles into clusters. Clusters of activities, artifacts, and roles were then named as key activities, artifacts, and roles. We identified seven key activities (i.e., security auditing, security analysis and testing, security training, security prioritization and monitoring, risk management, security planning and threat modeling; and security requirements engineering), five key artifacts (i.e., security requirement artifacts, security repositories, security reports, security tags, and security policies), and four key roles (i.e., security guru, security developer, penetration tester, and security team) in AESS. The identified key activities, artifacts, and roles can be used by software development teams to improve their software engineering processes in terms of software security.

Keywords: secure software development; security engineering; agile methods; agile development; software development; software engineering; software security; application security; cybersecurity; cyber resilience

Citation: Mihelič, A.; Hovelja, T.; Vrhovec, S. Identifying Key Activities, Artifacts and Roles in Agile Engineering of Secure Software with Hierarchical Clustering. *Appl. Sci.* **2023**, *13*, 4563. https://doi.org/10.3390/app13074563

Academic Editors: Howon Kim and Thi-Thu-Huong Le

Received: 27 January 2023
Revised: 24 March 2023
Accepted: 29 March 2023
Published: 4 April 2023

1. Introduction

When creating software, software development businesses adhere to a particular methodology, whether it is a widely recognized cookie-cutter software development methodology or a unique internal methodology that is only loosely defined by themselves. The process of producing software is made more controllable and efficient by employing a software development method, which entails using the appropriate practices, tools, and methodologies [1]. For any software development company attempting to generate a competitive product, an appropriate software development methodology is consequently essential. Among those, agile software development methods prevail in the current market [2]. Contrary to the traditional methods, most agile methods rely on short feedback loops with customers, self-organizing teams, daily face-to-face communications among team members, and iterative and incremental approaches to software development [3,4]. Such work organization can therefore realize the principles from the agile manifesto. Hence, agile methods provide a highly adaptive, effective, and quick software development process.

However, due to their loose design and prioritization of individuals, interactions, collaboration, and flexibility [5], agile methods tend to be less suitable for secure software development [6]. Agile engineering of secure software (AESS) has been gaining prominence during the last few years due to both the popularity of agile methods in practice and the need for engineering secure software in global cyberspace [7,8]. The AESS literature

appears to be blooming with a variety of AESS approaches being proposed and studied. Some of the proposed approaches represent comprehensive solutions in the form of a complete method or an adaptation of an existing one (e.g., [9–11]), though most studies propose partial solutions that can be integrated into several agile (or traditional) software development practices. However, several similarities in the proposed security solutions can be observed. For example, variations in security testing [12–14], risk analysis [15–17], threat modeling [18–20], and security-related backlogs [21–23] can be found in the literature, just to name a few. Hence, there is a benefit of consolidating all proposed security elements found in the AESS literature to gain insights into the variety of the proposed solutions and their elements and set the grounds for the future development of security solutions by exposing the knowledge gaps and possible over-saturation of particular security elements.

The paper is structured as follows. The next section presents a theoretical background. First, it presents the standards behind software development and then the security solutions found in the literature. The Section 3 presents the motivation behind the paper and its aim. The Section 3 presents the methodological framework in which the qualitative and quantitative methods were used to achieve the aim of the paper. The study results are presented in the Section 4, and the results are discussed in the Section 5. Finally, with the concluding remarks, we highlight the limitations of our work and give suggestions for future work.

2. Theoretical Background

The software development method typically defines the workflow of the software development process. A project management method is usually used to divide processes which can be grouped into distinct phases, making the complete process more manageable and enabling more efficient software production [24]. In software development, a conceptual framework considers the structure of the stages involved in the development of an application from its initial feasibility study through the deployment and maintenance, which is defined as a software development life-cycle (SDLC) [25]. More precisely, it is defined as a set of "processes, activities, and tasks that are to be applied during the acquisition of a software product or service and during the supply, development, operation, maintenance, and disposal of software products" [26]. SDLC is also standardized by the International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), and the Institute of Electrical and Electronics Engineers (IEEE) in international standards, namely, ISO/IEC/IEEE 12207:2017 [27] and ISO/IEC/IEEE 24748-1:2018 [28].

The ISO/IEC/IEEE 12207:2017 focuses on software development processes and defines groups of processes, namely, Organizational Project-Enabling processes, Technical Management processes, and *Technical processes*. It contains processes, activities, and tasks that could be applied during any stage of the software life-cycle, i.e., from the supply to the disposal [27]. On the other hand, the ISO/IEC/IEEE 24748-1:2018 more specifically addresses software life-cycle concepts, models, stages, processes, process applications, key points of view, adaptation. and use in various domains [28]. The standards ISO/IEC/IEEE 12207:2017 and ISO/IEC/IEEE 24748-1:2018 are to be used in combination. Any software life-cycle progresses through stages in which the software system is conceptualized, developed, produced as a product or service (or both), utilized, supported, and finally retired [28]. Hence, according to the ISO/IEC/IEEE 24748-1:2018 standard, the software development life-cycle has five distinct stages: concept (identification of the needs and expectations; exploring concepts and viable solutions); development (refining the requirements, creating solution descriptions, building the software, and verifying/testing it); production (producing, inspecting, and testing software); utilization (operating the software to meet the needs); support (providing sustainability of software capability); retirement (storing, archiving, or disposing the software). Each stage should have *entry* and *exit* criteria defined in advance. While defining such criteria, it is crucial to consider that the progressions between mentioned stages are not necessarily non-iterative and continuous. Iterations and

recursions are common at most life-cycle stages, making every software's life-cycle unique with only fundamental stages in common. However, the software development life-cycle is only a conceptual framework with the structure of the stages involved in the development. Actual approaches to SDLC, i.e., the SDLC models, are typically used to describe the steps within the life-cycle framework [25]. They can be broadly grouped into *traditional*, *agile*, or *hybrid* categories [29,30].

However, also according to the established international standards [31], software should be adequately secure. It should be able to resist alterations during intentional or unintentional attacks adequately. Hence, secure software development solutions emerged as a response to market needs and to meet the minimum standards of security. Since security is typically considered a non-functional requirement, it is frequently not integral to general cookie-cutter methods. Solutions for secure software development are generally a variation of "security add-on" for the existing methods.

In recent years, for example, AESS authors focused mainly on security prioritization through additional integrating security meetings into the development [32,33] and through recommendations for security experts stemming from categories identified in practice [8] and managing security requirements [6,34,35]. Furthermore, since user stories can be considered the driving force of AESS, security-related stories (a set of possible scenarios and threats to the end-product), such as safety stories [9], abuser stories [36], misuse and abuse cases [37], and generic security user stories [38], are suggested in the literature. Related to these, some authors additionally propose using security-specific backlogs that may provide a superior overview of security issues and requirements [21,36]. Others, however, suggest taking a different approach—by measuring and elevating the knowledge [39] and motivation of the development team as the baseline for their technical work. Gamification of security is one such approach. Based on planning poker [40] (heavily influenced by the wideband Delphi method), protection poker [41] and threat poker [42] were proposed as additional elements to be incorporated to the AESS. These aim to help determine the ease of executing the attack and the potential security/privacy impact. On the other hand, authors frequently propose including at least one additional security-specific role. Such roles typically include some variation of a security expert [43], such as a security master [21], security engineer [36], or security champion [44]. Even though these roles are among more common ones, there are also some others, such as security architect [10], penetration tester [14], and security developer [19]. Their primary role is to provide security knowledge and support to the development team.

Different AESS approaches often have their own focuses, application levels and characteristics [45–47]. Nevertheless, certain ideas seem to be reoccurring, and considerable overlaps between different AESS approaches can be noticed. Due to their fundamental differences, it is not an easy task to compare different AESS approaches, and it is much more complex to consolidate them. AESS approaches can, however, be further decomposed. Most AESS approaches involve a set of different activities, such as security intention recap meetings [33], that may be integrated into a software engineering method used in a certain software development enterprise. They often prescribe various artifacts, such as security backlog [22], which may be used during software engineering to enhance security of the developed software. Some AESS approaches also introduce roles, such as the security master [20], that may execute AESS activities and take advantage of AESS artifacts. Similarly to AESS approaches, considerable overlaps may be noticed between different activities, artifacts, and roles found in AESS approaches. However, they may be more directly compared than varying approaches due to their more atomic nature.

3. Motivation and Aim

Software engineering falls within the more applied sciences. After all, engineering is by its definition an "application of scientific and mathematical principles to practical ends" [48]. Hence, the results of the studies should be easy to interpret and implement into practice. Even though software engineering research has positively affected software

engineering practice [49], some authors note that industrial practitioners, even if interested in implementing ideas published in high-ranked scientific journals, have great difficulties, or it is nearly impossible for them to apply such approaches in practice [50]. Such a situation emerged for several reasons. Software engineering research must address several aspects to meet the needs of the industry. For example, it must focus on real-world requirements and constraints and address scalability. Moreover, the proposed approaches are continuously exposed to various human factors, and finally, the end product must meet the right trade-offs (e.g., between quality and cost) [51]. Additionally, most solutions proposed in the majority of academic papers, especially those in high-ranked journals, typically do not consider practical factors which are essential for real-world applications of solutions [50].

As previously mentioned, fundamental ideas in secure software engineering frequently overlap, e.g., gamification of security (e.g., [41,42]), security-themed requirement repositories (e.g., [21,36]), additional meetings [32,33], and various prioritization activities [32,34], among others. Hence, there is a need to consolidate all the security elements proposed in the literature for practitioners to have an overview of proposed solutions in a simplified form, thereby enabling a simplified overview of security-related solutions in the literature. When consolidated, security elements can be used as pieces of a jigsaw puzzle for tailoring software engineering processes in enterprises, focusing on the security of the developed software.

Therefore, we aimed to consolidate activities, artifacts, and roles, and thus identify the overall key activities, artifacts, and roles found in the literature on AESS. We first extensively reviewed the literature to gain initial sets of activities, artifacts, and roles. Next, we cross-evaluated them with similarity matrices. The similarity matrices were then converted into distance matrices, enabling the use of Ward's hierarchical clustering method. Finally, we consolidated activities, artifacts, and roles into clusters of activities, artifacts, and roles. These are named as the key AESS activities, artifacts, and roles.

4. Materials and Methods

To identify AESS activities, artifacts, and roles found in the literature, we conducted an extensive literature review covering conference and journal papers indexed in four bibliographic databases (i.e., ACM DL, IEEE Xplore, Scopus, and Web of Science). The search queries included a combination of keywords: agile, lean, secure, security, software, development, engineering, method, and management. The queries were used to search titles, abstracts, and keywords of bibliographic records published since 2000. Papers describing or testing a secure software development method compatible with agile methods were retained in the pool of relevant papers. Additionally, the snowball method [52] was employed to widen the reach of the literature review. During this process, we screened the references of initially identified papers for potentially relevant papers. Further, we backward-snowballed the papers with Google Scholar (i.e., we searched for papers citing the identified papers). The papers in the pool of relevant papers were carefully examined to identify the activities, artifacts, and roles they describe. The original names of activities, artifacts, and roles were preserved at this point, even though they may not be indicative of their types. For example, generic security user stories refer to an activity rather than an artifact, as the name suggests in [38]. One researcher extracted the data, and another researcher double-checked the extraction. To make sure that only activities, artifacts, and roles directly contributing to the security of developed software are included in the study, two researchers independently classified them as either related or non-related to security. The researchers disagreed in 11.3 percent of cases. The Cohen's kappa ($\kappa = 0.72, p < 0.001$) suggests substantial agreement between the researcher's assessments [53]. Classification inconsistencies were solved with consensus before moving on to the next step.

To determine similarities among the identified activities, artifacts, and roles, similarity matrices were employed. Similarity matrices are $n \times n$ tables of binary or multi-value scores indicating the likeness between two data points (e.g., activities). They can be used for reducing data in larger qualitative datasets [54]. This enabled us to consolidate

activities, artifacts, and roles. The consolidation was performed in three steps. First, two researchers independently created separate similarity matrices for activities, artifacts, and roles. Similarities were scored on a 3-point scale: 0.0 (not similar), 0.5 (partially similar), or 1.0 (very similar). The cross-evaluations of identified activities, artifacts, and roles (e.g., comparing all identified activities with each other) resulted in 3419 assessments by each researcher. The researchers assigned different similarity scores in 1.6 percent of cases. The Cohen's kappa ($\kappa = 0.96$, $p < 0.001$) suggests very strong (almost perfect) agreement between the researchers' assessments [53]. Such a strong agreement may be attributed to a large number of obvious dissimilarities among activities, artifacts, and roles. Second, similarity matrices produced by each researcher were consolidated into final similarity matrices by calculating the means of individual scores for each pair of security elements. Similarities in the final similarity matrices were therefore scored on a 5-point scale: 0.00 (not similar), 0.25, 0.50, 0.75, 1.00 (very similar).

To consolidate activities, artifacts, and roles, we employed Ward's hierarchical clustering. Since Ward's method requires distance matrices, we transformed the final similarity matrices into distance matrices by calculating $x_{distance} = 1 - x_{similarity}$. The final number of clusters, their naming, and brief descriptions were determined in a focus group of three researchers. The number of clusters for each type of security elements was determined by minimizing the number of clusters while retaining the similarity of intra-cluster security elements. Clusters were named either according to the most representative and typical security elements within the cluster (e.g., threat requirement map, security backlog, security requirements repository, and safety product backlog were grouped into the security repositories cluster) or by determining the umbrella term that best describes all elements in a particular cluster (e.g., security test cases review report, test phase code review report, security mechanism review report, security testing report, and security audit report were grouped into the security reports cluster). Descriptions of the clusters were based on the definitions of the security elements found in their respective papers.

5. Results

The literature review yielded 27 AESS approaches that were decomposed into 76 unique activities, 28 unique artifacts, and 8 unique roles. These are presented in Table 1, together with their respective sources. Several similarities among the identified activities, artifacts, and roles can be noticed, supporting the need to consolidate them.

Table 1. Activities, artifacts, and roles found in the literature.

Source	Activities	Artifacts	Roles
[17]	Risk analysis; Vulnerability analysis; Inventory risks; Security test; Security controls	Security user stories	-
[16]	Risk assessment; Prioritization of security requirements	Threat-requirement map; Security requirements repository	-
[55]	Security survey; Security triage	-	-
[12]	Risk identification; Design inspection; Risk-based security tests; Code inspection; Penetration testing; Risk analysis	Misuse cases; Attack tree	Security master
[10]	Risk analysis; Risk estimations	-	Security manager; Security architect; Security master; Penetration tester
[14]	Agile risk analysis; Security requirement analysis; Security planning; Pair penetration testing; Dynamic code analysis; Code review; Penetration testing	Security-related user stories	Penetration tester

Table 1. *Cont.*

Source	Activities	Artifacts	Roles
[56]	Defining security requirements; Threat modeling; Risk analysis; Use of static analysis tools; Code review; Security testing; Fuzz testing; Security review	Secure coding policies; Secure testing policies; Secure design; Security keywords	-
[36]	Identification of security sensitive assets; Formulation of abuser stories; Abuser story risk assessment; Abuser story and user story negotiation; Definition of security-related user stories; Abuser story–countermeasure cross-checking	Security-related user stories (security functionalities); Abuser stories (threat scenarios); Security related coding standards	-
[13]	Security requirements analysis and planning; Threat modeling and designing; Secure code implementation; Security testing; Security training	Abuser stories; Security user stories; Attack trees	-
[57]	Risk analysis	-	-
[58]	Utilization of highly testable architecture extensive testing; Security refactoring; Security test cases	-	-
[59]	Basic security training for all stakeholders	Fundamental security architecture	-
[15,23]	Hazard analysis; Risk analysis	Safety product backlog	-
[33]	Security intention recap meetings	-	-
[60]	Continuous risk management; Identification of business and technical risks; Synthesizing and prioritization of the risks; Code analysis	-	-
[18]	Threat modeling; Dynamic code analysis	-	Security team; Security champion
[19]	Security training; Additional security training after change; Application risk analysis; Test plan review; Threat modeling; Threat modeling updates; Business impact analysis; Security auditing; Security testing; Attack surface recognition and reduction; Security test cases review; Test phase code review; Use of automated testing tools (fuzzers & code analyzers); Security mechanism review; Development time auditing; Goal and criticality definition; Application security settings definition for maintenance	Documentation of security solutions; Security test plan; Threat models; Security testing report; Security audit report; Architecture security requirement; Attack surface analysis; Security mechanism review report; External interface review report; Test phase code review report; Security test case review report; Development-time audit report	Security developer
[20]	Vulnerability assessment; Threat modeling; Penetration testing; Code analysis	-	Security master; Security guru
[61,62]	Threat modeling; Risk estimation; Security goals identification; External review of the assurance case; Automated security tests and analysis	Security assurance cases; Security user stories	-
[21,22]	-	Security backlog	-
[4]	Identification of security issues; Security implementation; Verification of the software from a security perspective	S-tags; S-marks	-
[41]	Protection poker	-	-
[42]	Threat poker	-	-
[37]	Security analysis; Design the security requirements; Security testing	Misuse cases; Abuse cases	-
[63]	Security analysis; Security modeling; Security designing; Security testing	-	-
[38]	Generic security user stories	-	-

Overall, the hierarchical clustering produced 16 clusters. Figure 1 shows a dendogram of how unique activities were grouped into seven clusters. *Security auditing* includes seven unique activities, such as security auditing, design inspection, and code review, that help to evaluate the overall security of developed software. *Security analysis and testing* is a set of 16 activities, such as penetration testing and risk-based security tests, aiming to analyze and test the security performance of developed software. *Security training* is a cluster of three activities aiming to increase the knowledge about security among software developers and other stakeholders. *Security prioritization and monitoring* is a set of 28 unique activities, such as protection poker and business impact analysis, that prioritize specific security goals and requirements, and monitor their implementation. *Risk management* is a cluster of six activities, such as hazard analysis, risk identification, and risk assessment, that help to identify, assess, and control threats to developed software on a strategic level. *Security planning and threat modeling* is a set of six activities, such as threat modeling and security modeling, for structured identification of potential operational threats (e.g., structural vulnerabilities and the absence of appropriate safeguards). *Security requirements engineering* is a group of 10 activities, such as defining security requirements, defining security-related user stories, and formulation of abuser stories, that focus on defining, documenting, and maintaining security requirements during software design.

Artifacts were grouped into five clusters, as shown in Figure 2. *Security requirement artifacts* is a cluster of seven backlog-related artifacts, such as abuse cases and security user stories, that put a focus on security during acquisition of requirements. *Security repositories* is a group of four artifacts, such as security requirements repository and security backlog, that provide checklists of requirements for software security. *Security reports* is a set of five artifacts, such as security audit reports and security testing reports. These are documents written at the end of certain security activities. *Security tags* is a cluster of three artifacts, such as security keywords and s-tags, that help developers to stay aware of the security relevance of users' stories. They point to parts of the developed software that need security verification. *Security policies* is a group of nine artifacts, such as secure coding policies and security test plans, that help to plan software development by providing standards and solutions related to security.

Figure 3 presents four clusters into which unique roles were grouped. *Security guru* is a cluster of four roles, such as security manager and security master, that are respected for their security knowledge and who lead software engineering from the security perspective on the strategic level. *Security developer* is a group of two roles, namely, security architect and security developer, that involve designing security tests, risk analyses, threat models, and attack surface analysis. *Penetration tester* is a cluster with a single role responsible for testing developed software for known vulnerabilities and attack vectors. He also tries to find possible exploits in the developed software. *Security team* is a cluster with a single team of several security-related roles, such as penetration tester, security manager, and security architect.

The above presented clusters may be considered as consolidated key AESS activities, artifacts, and roles, since they are composed of elements that are most similar to each other. The focus group was also able to provide descriptions of clusters that encompassed all their individual elements.

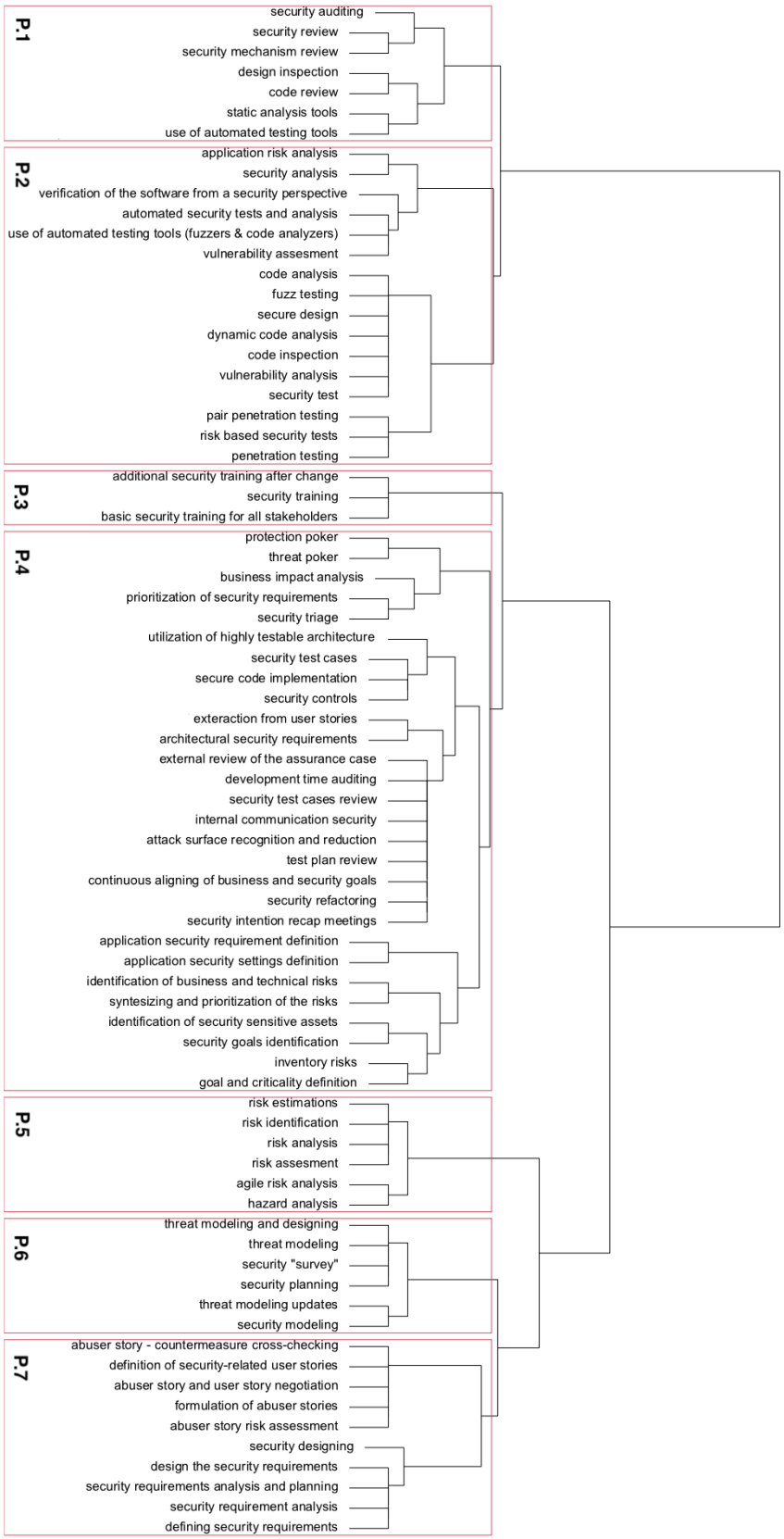

Figure 1. Dendrogram of activities. It indicates seven distinct clusters: *security auditing* (P.1), *security analysis and testing* (P.2), *security training* (P.3), *security prioritization and monitoring* (P.4), *risk management* (P.5), *security planning and threat modeling* (P.6), and *security requirements engineering* (P.7).

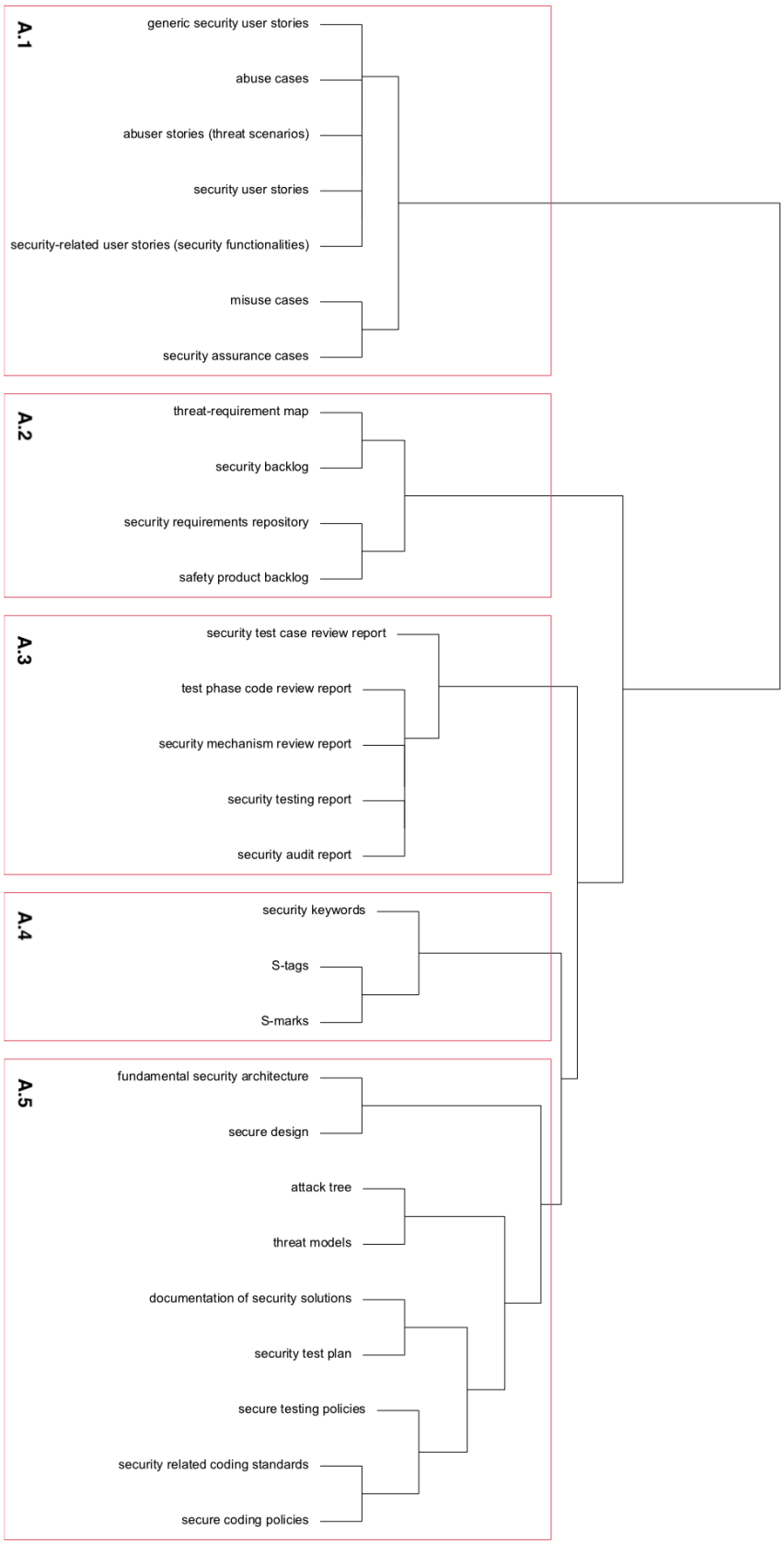

Figure 2. Dendrogram of artifacts. It indicates five distinct clusters: *security requirement artifacts* (A.1), *security repositories* (A.2), *security reports* (A.3), *security tags* (A.4), and *security policies* (A.5).

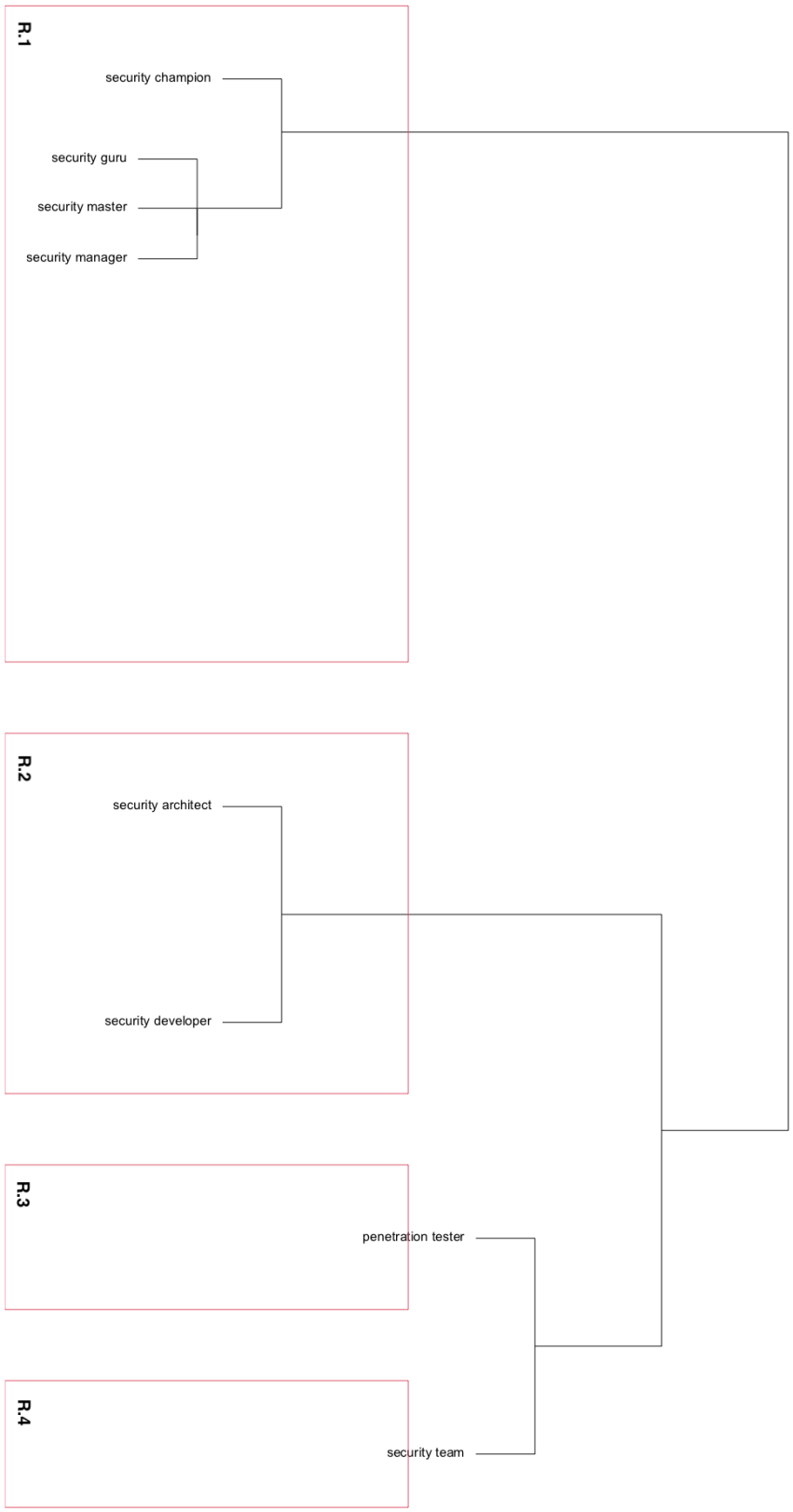

Figure 3. Dendrogram of roles. It indicates four distinct clusters: *security guru* (R.1), *security developer* (R.2), *penetration tester* (R.3), and *security team* (R.4).

6. Discussion

This paper aimed to offer an overview of security solutions proposed in the literature by disassembling proposed AESS approaches and methods into atomic parts (i.e., security elements) and consolidating the overlapping ideas. The consolidation process was performed using a mixed-methods approach, combining qualitative and quantitative methods. The consolidated security elements (i.e., clusters) can then be used by the practitioners as general guidelines that are presented concisely and interpretably while still allowing distinct elements to be traced back to their sources, as presented in Table 1. On the other hand, such an overview can also be used by researchers when analyzing knowledge gaps and designing future research. Hence, this paper yields several theoretical and practical implications.

First, this study comprehensively reviews the literature on AESS. Although some papers reviewed AESS (e.g., [7,64,65]), this is among the first studies to comprehensively amass a number of AESS approaches and decompose them into corresponding activities, artifacts, and roles. Such a generalized overview of the proposed security solutions should help practitioners and researchers understand the current state of the research on AESS. More particularly, our research offers insights into the available security elements and exposes the ones that should have received less attention, indicating research gaps.

Second, the research on AESS topics is diverse, and at first glance, individual studies and/or security solutions are seemingly not directly comparable. The clustering results presented in this study introduce the baseline, by which present (and potential future) research and security solutions can be compared. The comparisons can be made among the security solutions where security elements are categorized within the same clusters.

Third, this research gave an overview of AESS approaches and their constituent activities, artifacts, and roles. This list can aid practitioners looking for established, off-the-shelf AESS approaches. Although implementing an off-the-shelf AESS approach may not work well in specific settings, it may still offer a good starting point for software enterprises looking to upgrade their software engineering process, for example, for transitioning from near-zero security "as agile as it gets" software engineering towards more security-focused development of software. Evaluation of security elements independently of each other allows enterprises to modify their existing method only slightly by including only one most suitable element for their needs. The needs may vary from project to project. For example, some projects may require the highest levels of security, and others may call only for basic security requirements. Such an overview of security elements, as presented in this paper, where most security elements found in the literature could be placed in one of the clusters, offers decision-makers and agile teams a comprehensive list to choose from. Whether they seek low-hanging fruits or comprehensive security solutions, our results can serve as a checklist for enterprises when attempting to improve their development methods in terms of security.

Additionally, the clusters reveal the variety and the range of security elements the development enterprises can use. For example, if the enterprise aims to implement a security review in its development process, our results enable them to gain insights into the variety of security review measures and several different possibilities at their disposal.

Fourth, this study is one of the first to attempts at the qualitative-quantitative consolidation of the multitude of different activities, artifacts, and roles related to AESS found in the current literature. The identified clusters resemble the core engineering phases of secure software proposed by [66]: software requirements security, software design security, software construction security, and software testing security. However, this paper provides more insights by decomposing several AESS approaches into more atomic parts, including activities, artifacts, and roles. Several previously proposed secure software development frameworks, approaches, and maturity models, such as [67–70], provide some support for the identified key activities, artifacts, and roles. Therefore, clusters of activities, artifacts, and roles may be considered key AESS activities, artifacts, and roles. They can be used as pieces of a jigsaw puzzle for tailoring software engineering processes in enterprises with a

focus on the security of the developed software. Future studies may also consider focusing on the key AESS activities, artifacts, and roles instead of specific AESS approaches.

The present study has some limitations that the reader should note. First, the literature review was conducted with four bibliographic databases (Web of Science, Scopus, ACM Digital Library, and IEEE Xplore) with a search query that resulted in the most comprehensive overview of published documents while maintaining the manageability of the results. Hence, some of the proposed approaches may not be included in our review. To address the issue, we additionally employed the snowball and backward-snowball methods to identify additional studies we might have missed. Additionally, this study could have omitted certain relevant studies due to circumstances beyond our control (e.g., studies published in languages other than English, and non-availability of papers). Second, since we focused on agile, elements from traditional methods also relevant for agile secure software engineering may have been absent. Third, the similarity matrices included a vast number of security elements that needed to be compared. Since the comparisons were made by the researchers based on the descriptions in the respective papers (some elements were described in more detail, whereas others remained scarce with the description), some of the comparisons may be subject to errors. To reduce the subjectivity of the results, the dimensionality reduction was conducted by analyzing the input data qualitatively and quantitatively, with the help of several researchers. However, we mitigated the issue by following the protocol: two researchers independently created similarity matrices. Due to the possibility of researcher bias in creating similarity matrices, Ward's hierarchical clustering was performed to suggest the ideal number and content of clusters, and the focus group of three researchers confirmed the clusters.

Future studies may incorporate such potentially undetected AESS approaches into the identified clusters, or identify new ones. Second, the study focused on agile engineering of secure software. Future work may investigate activities, artifacts, and roles beyond those already known to be applicable to agile methods. For example, future studies may investigate ideas from traditional engineering of secure software to determine if and how they perform when used in agile methods. Such studies may be especially beneficial in identifying prospective new AESS activities, artifacts, and roles. Third, similarity matrices were created by the researchers, which introduces the possibility of researcher bias. We aimed to lower the possibility of researcher bias by first having two researchers independently create the similarity matrices and then calculating the means of their scores for each pair of security elements. Despite our best efforts to minimize it, the possibility of researcher bias was not zero. Future studies may incorporate natural language processing to automatize this process, for example, by determining similarities between activities, artifacts, and roles based on their descriptions. Finally, future studies may investigate the impacts, such as costs, the agility of software development, and the security of the developed software, when incorporating individual key activities, artifacts, and roles in software development processes.

7. Conclusions

This paper presented the identification and clustering of security elements found in the AESS literature. We reviewed the literature to gain initial sets of activities, artifacts, and roles. Activities, artifacts, and roles were then cross-evaluated with similarity matrices. Based on these matrices, we performed Ward's hierarchical clustering. We identified seven key activities (i.e., security auditing, security analysis and testing, security training, security prioritization and monitoring, risk management, security planning and threat modeling, and security requirement engineering), five key artifacts (i.e., security requirement artifacts, security repositories, security reports, security tags, and security policies), and four key roles (i.e., security guru, security developer, penetration tester, and security team) in AESS. The software development teams can leverage these key security elements at any core engineering phase of software development (depending on which security element they shall choose to implement into their process and where within the development

process they would like to introduce more security-centered elements). Depending on the nature of the agile teams and the levels of agility their development processes are built upon, they may choose only the security elements that fit best into their development processes. Furthermore, researchers can use the insights from this paper to identify possible knowledge gaps and propose new security solutions that can be applied to AESS practice.

Author Contributions: Conceptualization, A.M., T.H. and S.V.; methodology, A.M.; validation, S.V.; formal analysis, A.M.; resources, A.M.; investigation, A.M., T.H. and S.V.; data curation, A.M.; writing—original draft preparation, A.M. and S.V.; writing—review and editing, A.M., T.H. and S.V.; visualization, A.M.; supervision, T.H. and S.V.; funding acquisition, S.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

AESS	Agile engineering of secure software

References

1. Bianchi, M.J.; Conforto, E.C.; Amaral, D.C. Beyond the agile methods: A diagnostic tool to support the development of hybrid models. *Int. J. Manag. Proj. Bus.* **2021**, *14*, 1219–1244. [CrossRef]
2. Rindell, K.; Ruohonen, J.; Holvitie, J.; Hyrynsalmi, S.; Leppänen, V. Security in agile software development: A practitioner survey. *Inf. Softw. Technol.* **2021**, *131*, 106488. [CrossRef]
3. Adelyar, S.H.; Norta, A. Towards a Secure Agile Software Development Process. In Proceedings of the 10th International Conference on the Quality of Information and Communications Technology (QUATIC), Lisbon, Portugal, 6–9 September 2016; pp. 101–106. [CrossRef]
4. Pohl, C.; Hof, H.J. Secure Scrum: Development of Secure Software with Scrum. In Proceedings of the The Ninth International Conference on Emerging Security Information, Systems and Technologies Secure, Venice, Italy, 23–28 August 2015; pp. 15–20.
5. Beck, K.; Beedle, M.; van Bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; et al. Manifesto for Agile Software Development. 2001.
6. Tøndel, I.A.; Jaatun, M.G. Towards a Conceptual Framework for Security Requirements Work in Agile Software Development. *Int. J. Syst. Softw. Secur. Prot.* **2020**, *11*, 33–62. [CrossRef]
7. Tashtoush, Y.M.; Darweesh, D.A.; Husari, G.; Darwish, O.A.; Darwish, Y.; Issa, L.B.; Ashqar, H.I. Agile Approaches for Cybersecurity Systems, IoT and Intelligent Transportation. *IEEE Access* **2022**, *10*, 1360–1375. [CrossRef]
8. Tøndel, I.A.; Cruzes, D.S.; Jaatun, M.G.; Sindre, G. Influencing the security prioritisation of an agile software development project. *Comput. Secur.* **2022**, *118*, 102744. [CrossRef]
9. Barbareschi, M.; Barone, S.; Carbone, R.; Casola, V. Scrum for safety: An agile methodology for safety-critical software systems. *Softw. Qual. J.* **2022**, *30*, 1067–1088. [CrossRef]
10. Baca, D.; Boldt, M.; Carlsson, B.; Jacobsson, A. A Novel Security-Enhanced Agile Software Development Process Applied in an Industrial Setting. In Proceedings of the ARES Conference International Conference on Availability, Reliability and Security 2015, Toulouse, France, 24–28 August 2015; pp. 11–19. [CrossRef]
11. Alenezi, M.; Basit, H.A.; Beg, M.A.; Shaukat, M.S. Synthesizing secure software development activities for linear and agile lifecycle models. *Software: Pract. Exp.* **2022**, *52*, 1426–1453. [CrossRef]
12. Firdaus, A.; Ghani, I.; Jeong, S.R. Secure Feature Driven Development (SFDD) Model for Secure Software Development. *Procedia-Soc. Behav. Sci.* **2014**, *129*, 546–553. [CrossRef]
13. Singhal, S.; Singhal, A. Development of Agile Security Framework Using a Hybrid Technique for Requirements Elicitation. In *Advances in Computing, Communication and Control*; Unnikrishnan, S., Surve, S., Bhoir, D., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 178–188.

14. Maier, P.; Ma, Z.; Bloem, R. Towards a Secure SCRUM Process for Agile Web Application Development. In Proceedings of the 12th International Conference on Availability, Reliability and Security—ARES '17, Reggio Calabria, Italy, 29 August–1 September 2017; pp. 1–8. [CrossRef]
15. Stålhane, T.; Johnsen, S.O. Resilience and safety in agile development (Through safescrum). In Proceedings of the 27th European Safety and Reliability Conference, ESREL 2017, Portoroz, Slovenia, 18–22 June 2017; pp. 945–954. [CrossRef]
16. Ionita, D.; Van Der Velden, C.; Ikkink, H.j.K.; Eelko, N. Towards Risk-Driven Security Requirements Management in Agile Software Development. *Lect. Notes Bus. Inf. Process.* **2019**, *350*, 133–144. [CrossRef]
17. Maria, R.E.; Rodrigues, L.A.; Pinto, N.A. ScrumS—A model for safe agile development. In Proceedings of the 7th International ACM Conference on Management of Computational and CollEctive Intelligence in Digital EcoSystems, MEDES 2015, Caraguatatuba, Brazil, 25–29 October 2015; pp. 43–47. [CrossRef]
18. Nguyen, J.; Dupuis, M. Closing the feedback loop between UX design, software development, security engineering, and operations. In Proceedings of the 20th Annual Conference on Information Technology Education—SIGITE 2019, Tacoma, WA, USA, 3–5 October 2019; pp. 93–98. [CrossRef]
19. Rindell, K.; Hyrynsalmi, S.; Leppänen, V. Securing scrum for VAHTI. In Proceedings of the CEUR Workshop Proceedings, Paris, France, 31 August–3 September 2015; pp. 236–250.
20. Mohino, J.d.V.; Higuera, J.B.; Higuera, J.R.B.; Montalvo, J.A.S. The application of a new secure software development life cycle (S-SDLC) with agile methodologies. *Electronics* **2019**, *8*, 1218. [CrossRef]
21. Azham, Z.; Ghani, I.; Ithnin, N. Security backlog in scrum security practices. In Proceedings of the 5th Malaysian Conference in Software Engineering, MySEC 2011, Johor Bahru, Malaysia, 13–14 December 2011; pp. 414–417. [CrossRef]
22. Ghani, I.; Azham, Z.; Jeong, S.R. Integrating software security into agile-Scrum method. *KSII Trans. Internet Inf. Syst.* **2014**, *8*, 646–663. [CrossRef]
23. Stålhane, T.; Myklebust, T.; Hanssen, G. The application of safe scrum to IEC 61508 certifiable software. In Proceedings of the 11th International Probabilistic Safety Assessment and Management Conference and the Annual European Safety and Reliability Conference 2012, Helsinki, Finland, 25–29 June 2012; pp. 6052–6061.
24. Project Management Institute. *A guide to the project management body of knowledge (PMBOK guide)*, 6th ed.; Project Management Institute: Newtown Square, PA, USA, 2017; p. 579.
25. Ruparelia, N.B. Software development lifecycle models. *ACM SIGSOFT Softw. Eng. Notes* **2010**, *35*, 8–13. [CrossRef]
26. *ISO/IEC 12207:2008*; Systems and Software Engineering—Software Life Cycle Processes. International Organization for Standardization: Geneva, Switzerland, 2008.
27. *ISO/IEC/IEEE 12207:2017(E)*, 1st ed.; Systems and Software Engineering—Software Life Cycle Processes. ISO/IEC/IEEE International Standard: Geneva, Switzerland, 2017; pp. 1–157. [CrossRef]
28. *ISO/IEC/IEEE 24748-1:2018(E)*; Systems and Software Engineering—Life Cycle Management—Part 1: Guidelines for Life Cycle Management. ISO/IEC/IEEE International Standard: Geneva, Switzerland, 2018; pp. 1–82. [CrossRef]
29. Kuhrmann, M.; Diebold, P.; Munch, J.; Tell, P.; Trektere, K.; McCaffery, F.; Garousi, V.; Felderer, M.; Linssen, O.; Hanser, E.; et al. Hybrid Software Development Approaches in Practice: A European Perspective. *IEEE Softw.* **2019**, *36*, 20–31. [CrossRef]
30. Gemino, A.; Horner Reich, B.; Serrador, P.M. Agile, Traditional, and Hybrid Approaches to Project Success: Is Hybrid a Poor Second Choice? *Proj. Manag. J.* **2021**, *52*, 161–175. [CrossRef]
31. *ISO/IEC 15408-1:2009*; Information Technology—Security Techniques—Evaluation Criteria for IT Security—Part 1: Introduction and General Model. ISO: Geneva, Switzerland, 2009.
32. Tøndel, I.A.; Cruzes, D.S. Continuous software security through security prioritisation meetings. *J. Syst. Softw.* **2022**, *194*, 111477. [CrossRef]
33. Tøndel, I.A.; Cruzes, D.S.; Jaatun, M.G.; Rindell, K. The Security Intention Meeting Series as a way to increase visibility of software security decisions in agile development projects. In Proceedings of the International Conference on Availability, Reliability and Security, Canterbury, Canterbury, UK, 26–29 August 2019; pp. 1–8. [CrossRef]
34. Behutiye, W.; Rodriguez, P.; Oivo, M. Quality Requirement Documentation Guidelines for Agile Software Development. *IEEE Access* **2022**, *10*, 70154–70173. [CrossRef]
35. Reddivari, S. An Agile Framework for Security Requirements: A Preliminary Investigation. In Proceedings of the 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), IEEE, Los Alamitos, CA, USA, 27 June–1 July 2022; pp. 432–433. [CrossRef]
36. Boström, G.; Wäyrynen, J.; Bodén, M.; Beznosov, K.; Kruchten, P. Extending XP practices to support security requirements engineering. In Proceedings of the 2006 international workshop on Software engineering for secure systems—SESS '06, Shanghai, China, 20–21 May 2006; pp. 11–17. [CrossRef]
37. Daud, M.I. Secure software development model: A guide for secure software life cycle. In Proceedings of the International MultiConference of Engineers and Computer Scientists 2010, IMECS 2010, Hong Kong, 17–19 March 2010; pp. 724–728.
38. Siiskonen, T.; Sars, C.; Vah Sipila, A.; Pietikain, A. Generic Security User Stories. In *Handbook of the Secure Agile Software Development Life Cycle*; Pietikinen, P., Rning, J., Eds.; University of Oulu: Oulu, Finland, 2014; Chapter 9, pp. 9–14.
39. Oyetoyan, T.D.; Jaatun, M.G.G.; Cruzes, D.S. Measuring Developers' Software Security Skills, Usage, and Training Needs. In *Research Anthology on Agile Software, Software Development, and Testing*; IGI Global: Hershey, PA, USA, 2022; pp. 2026–2048. [CrossRef]

40. Grenning, J. Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods Renaiss. Softw. Consult.* **2002**, *3*, 22–23.
41. Williams, L.; Meneely, A. Protection poker: The New Software Security "Game". *IEEE Secur. Priv.* **2010**, *8*, 14–20. [CrossRef]
42. Rygge, H.; Jøsang, A. Threat Poker : Solving Security and Privacy Threats in Agile Software Development. In Proceedings of the 23rd Nordic Conference on Secure IT Systems, Oslo, Norway, 28–30 November 2018; pp. 1–15. [CrossRef]
43. Musa, S.B.; Norwawi, N.M.; Selamat, M.H.; Sharif, K.Y. Improved extreme programming methodology with inbuilt security. In Proceedings of the 2011 IEEE Symposium on Computers & Informatics, Kuala Lumpur, Malaysia, 20–23 March 2011; pp. 674–679. [CrossRef]
44. Jaatun, M.G.; Bernsmed, K.; Cruzes, D.S.; Tøndel, I.A. Threat Modeling in Agile Software Development. In *Research Anthology on Agile Software, Software Development, and Testing*; IGI Global: Hershey, PA, USA, 2022; pp. 480–490. [CrossRef]
45. Bernsmed, K.; Cruzes, D.S.; Jaatun, M.G.; Iovan, M. Adopting threat modelling in agile software development projects. *J. Syst. Softw.* **2022**, *183*, 111090. [CrossRef]
46. Althar, R.R.; Samanta, D.; Kaur, M.; Singh, D.; Lee, H.N. Automated Risk Management Based Software Security Vulnerabilities Management. *IEEE Access* **2022**, *10*, 90597–90608. [CrossRef]
47. Kumar, S.; Kaur, A.; Jolly, A.; Baz, M.; Cheikhrouhou, O. Cost Benefit Analysis of Incorporating Security and Evaluation of Its Effects on Various Phases of Agile Software Development. *Math. Probl. Eng.* **2021**, *2021*, 7837153. [CrossRef]
48. Parton, J. *The American Heritage Dictionary of the English Language*; Houghton Mifflin: Boston, MA, USA, 2022.
49. Osterweil, L.J.; Ghezzi, C.; Kramer, J.; Wolf, A.L. Determining the Impact of Software Engineering Research on Practice. *Compuer* **2008**, *41*, 39–49. [CrossRef]
50. Practical relevance of software engineering research: Synthesizing the community's voice. *Empir. Softw. Eng.* **2020**, *25*, 1687–1754. [CrossRef]
51. Embracing the engineering side of software engineering. *IEEE Softw.* **2012**, *29*, 96–99. [CrossRef]
52. Wohlin, C.; Prikladniki, R. Systematic literature reviews in software engineering. *Inf. Softw. Technol.* **2013**, *55*, 919–920. [CrossRef]
53. McHugh, M.L. Interrater reliability: The kappa statistic. *Biochem. Medica* **2012**, *22*, 276–282. [CrossRef]
54. Namey, E.; Guest, G.; Thairu, L.; Johnson, L. Data reduction techniques for large qualitative data sets. In *Handbook for Team-Based Qualitative Research*; AltaMira Press: Lanham, MD, USA, 2008; pp. 137–161.
55. Giacalone, M.; Paci, F.; Mammoliti, R.; Perugino, R.; Massacci, F.; Selli, C. Security Triage: An Industrial Case Study on the Effectiveness of a Lean Methodology to Identify Security Requirements. In Proceedings of the Symposium on Empirical Software Engineering and Measurement—ESEM 2014, Torino, Italy, 18–19 September 2014; pp. 1–8. [CrossRef]
56. Koc, G.; Aydos, M.; Tekerek, M. Evaluation of Trustworthy Scrum Employment for Agile Software Development based on the Views of Software Developers. In Proceedings of the UBMK 2019 4th International Conference on Computer Science and Engineering, Samsun, Turkey, 11–15 September 2019; pp. 63–67. [CrossRef]
57. Singh, A. Integrating the Extreme Programing Model with Secure Process for Requirement Selection. In Proceedings of the 2nd International Conference on Electronics, Communication and Aerospace Technology—ICECA 2018, Coimbatore, India, 29–31 March 2018; pp. 423–426. [CrossRef]
58. Tappenden, A.F.; Huynh, T.; Miller, J.; Geras, A.; Smith, M. Agile Development of Secure Web-Based Applications. *Int. J. Inf. Technol. Web Eng. (IJITWE)* **2006**, *1*, 1–24. [CrossRef]
59. Ge, X.; Paige, R.; Polack, F.; Brooke, P. Extreme Programming Security Practices. In *Agile Processes in Software Engineering and Extreme Programming, Proceedings of the 8th International Conference, XP 2007, Como, Italy, 18–22 June 2007*; Concas, G., Damiani, E., Scotto, M., Succi, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 226–230.
60. Yu, W.D.; Le, K. Towards a secure software development lifecycle with SQUARE+R. In Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference, Izmir, Turkey, 16–20 July 2012; pp. 565–570. [CrossRef]
61. Othmane, L.; Angin, L.; Weffers, H.; Bhargava, B. Extending the Agile Development Process to Develop Acceptably Secure Software. *IEEE Trans. Dependable Secur. Comput.* **2014**, *11*, 497–509. [CrossRef]
62. Othmane, L.B.; Angin, P.; Bhargava, B. Using assurance cases to develop iteratively security features using scrum. In Proceedings of the Proceedings—9th International Conference on Availability, Reliability and Security, ARES 2014, Fribourg, Switzerland, 8–12 September 2014; pp. 490–497. [CrossRef]
63. Mougouei, D.; Sani, N.F.M.; Almasi, M.M. S-Scrum : A Secure Methodology for Agile Development of Web Services. *World Comput. Sci. Inf. Technol. J. (WSCIT)* **2013**, *3*, 15–19.
64. López, L.; Burgués, X.; Martínez-Fernández, S.; Vollmer, A.M.; Behutiye, W.; Karhapää, P.; Franch, X.; Rodríguez, P.; Oivo, M. Quality measurement in agile and rapid software development: A systematic mapping. *J. Syst. Softw.* **2022**, *186*, 111187. [CrossRef]
65. Nägele, S.; Watzelt, J.P.; Matthes, F. Investigating the Current State of Security in Large-Scale Agile Development. In *Agile Processes in Software Engineering and Extreme Programming, Proceedings of the 23rd International Conference on Agile Software Development, XP 2022, Copenhagen, Denmark, 13–17 June 2022*; Lecture Notes in Business Information Processing; Springer: Cham, Switzerland, 2022; Volume 445, pp. 203–219. [CrossRef]
66. Nina, H.; Pow-Sang, J.A.; Villavicencio, M. Systematic mapping of the literature on Secure Software Development. *IEEE Access* **2021**, *9*, 36852–36867. [CrossRef]

67. Ansari, M.T.J.; Pandey, D.; Alenezi, M. STORE: Security Threat Oriented Requirements Engineering Methodology. *J. King Saud Univ.—Comput. Inf. Sci.* **2018**, *34*, 191–203. [CrossRef]
68. Karim, N.S.A.; Albuolayan, A.; Saba, T.; Rehman, A. The practice of secure software development in SDLC: An investigation through existing model and a case study. *Secur. Commun. Netw.* **2016**, *9*, 5333–5345. [CrossRef]
69. Migues, S.; Erlikhman, E.; Ewers, J.; Nassery, K. Building Security in Maturity Model (BSIMM) Foundations Report—Version 12.
70. Jaatun, M.G.; Soares Cruzes, D. Care and Feeding of Your Security Champion. In Proceedings of the 2021 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), Dublin, Ireland, 14–18 June 2021; pp. 1–7. [CrossRef]