Requirements Engineering (2020) 25:439–468
https://doi.org/10.1007/s00766-020-00338-w

ORIGINAL ARTICL

An efficient approach for reviewing security-related aspects in agile requirements specifications of web applications

Hugo Villamizar[1] · **Marcos Kalinowski**[1] · **Alessandro Garcia**[1] · **Daniel Mendez**[2]

Received: 11 December 2019 / Accepted: 1 September 2020 / Published online: 18 September 2020

Abstract
Defects in requirement specifications can have severe consequences during the software development life cycle. Some of them may result in poor product quality and/or time and budget overrun due to incorrect or missing quality characteristics, such as security. This characteristic requires special attention in web applications because they have become a target for manipulating sensible data. Several concerns make security difficult to deal with. For instance, security requirements are often misunderstood and improperly specified due to lack of security expertise and emphasis on security during early stages of software development. This often leads to unspecified or ill-defined security-related aspects. These concerns become even more challenging in agile contexts, where lightweight documentation is typically produced. To tackle this problem, we designed an approach for reviewing security-related aspects in agile requirements specifications of web applications. Our proposal considers user stories and security specifications as inputs and relates those user stories to security properties via natural language processing. Based on the related security properties, our approach identifies high-level security requirements from the Open Web Application Security Project (OWASP) to be verified and generates a reading technique to support reviewers in detecting defects. We evaluate our approach via three experimental trials conducted with 56 novice software engineers, measuring effectiveness, efficiency, usefulness and ease of use. We compare our approach against using: (1) the OWASP high-level security requirements and (2) a perspective-based approach as proposed in contemporary state of the art. The results strengthen our confidence that using our approach has a positive impact (with large effect size) on the performance of inspectors in terms of effectiveness and efficiency.

Keywords Agile requirements · Requirement verification · Software inspection · Software security

1 Introduction

Requirement engineering (RE) is an inherently complex part of software engineering. Given its complexity, defects such as ambiguities, inconsistencies and incomplete requirements may arise. These defects have been reported by practitioners to be causing problems in software projects, such as poor product quality and time and budget overruns [23]. Moreover, the costs for correcting these RE-related problems increase throughout the software development life cycle [9]. These additional costs reinforce the importance of identifying such defects at early stages.

The rapidly changing business environments in which many companies operate are challenging traditional RE approaches. This gave rise to agile methods for RE. Agile RE relies on lightweight documentation and face-to-face collaborations between customers and developers [10]. Yet, agility does not necessarily compensate for problems that have been reported for plan-driven software process models (e.g., RUP, V-Model XT, Waterfall), such as moving targets and communication flaws [23]. It can even make those problems more explicit if a critical prerequisite for successful RE are not met: human-intensive exchange, collaboration and

✉ Hugo Villamizar
hvillamizar@inf.puc-rio.br

Marcos Kalinowski
kalinowski@inf.puc-rio.br

Alessandro Garcia
afgarcia@inf.puc-rio.br

Daniel Mendez
daniel.mendez@bth.se

[1] Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil

[2] Blekinge Institute of Technology, Karlskrona, Sweden

trust [24]. In other words, agile RE has helped to address some RE problems, but it has also hindered others, such as under-specified requirements that are too abstract [23].

Security is an essential non-functional requirement (NFR) that requires special attention of any software system that manages valuable data and processes, among others, due to business needs to protect restricted user information. Much of sensitive information is hosted on the internet, making web applications an often favored target. In today's software systems, security vulnerabilities are increasing [16] and ensuring security controls is becoming more difficult. Defects in security requirements can lead to important vulnerabilities that impact core functionality, leading to loss of reputation, financial penalties and even legal consequences. For instance, in September 2016, the internet giant Yahoo announced it had been the victim of the biggest data breach in history [25]. The hackers stole personal information of 500 million users. The attack was performed via phishing emails with a link. Once it was clicked, malware was downloaded to the network. Thus, bad actors gained access to the user database. This can be seen as a breach not covered by the security requirements. Not surprisingly, security is considered as a relevant NFR for the twenty-first century [44].

For that reason, specifying and verifying security requirements is crucial to ensure software product quality. Nevertheless, security requirements are often misunderstood and improperly specified due to lack of security expertise and emphasis on security during early stages of software development [38]. While software requirement inspections represent a promising approach to effectively verify security requirements, security expertise is essential, but often lacking in software engineers [12]. Hence, software engineers could benefit from specific reading techniques to support the verification of security aspects during requirement inspections.

The picture is even more challenging in agile contexts. Most agile teams do not have a security expert on board. Therefore, product owners (POs) and developers are, at best, responsible for identifying and prioritizing security requirements [14]. However, POs as well as developers often lack security knowledge [52]. This may results in software which fails to properly deal with security. According to Eberlein [19], there is a need for agile methods to include techniques that make it possible to identify NFRs early. There is also a need to describe them in such a way that they may be analyzed early, thus reducing the likelihood of costly rework [36]. Alsaqaf et al. [1] conducted a literature review on engineering NFRs in agile projects that cover security concerns. They reported 12 challenges such as the inability of user stories, the most used artifact in agile RE, to document quality requirements, the product owner's lack of knowledge, the dependence on the product owner as the single point to collect the requirements and the delay in the validation of the requirements. That is why several recent secondary studies acknowledge the urgent need for methods to systematically engineer security requirements in agile projects [1, 56]. In that direction, Villamizar et al. [56] identified that most of the studies dealing with security in agile contexts lack empirical evaluation and research on requirement verification. Such activities should be conducted to assure those agile requirements specifications are correct, consistent, unambiguous and complete. This means appropriately covering basic security-related aspects, such as input validation, unauthorized access, assignment of administrative privileges and denial-of-service attacks.

Although these secondary studies have reported several challenges in the literature, the number of solution proposals to address these concerns is limited. Existing approaches (e.g., [12, 20, 43, 46]) employ inspection techniques to verifying security or identifying security goals from textual documents. These proposals are not focused on agile, but this does not mean that they cannot address these kinds of requirements. For instance, Carver et al. [12] propose a perspective-based approach to identify security defects in general requirements. In this case, the authors do not explicitly mention considering their approach in an agile context, but that does not exclude it to cover textual requirements or agile specifications. To the best of our knowledge, only one study explicitly proposes a methodology to address security verification activities in agile software development [18]. The study proposes a lightweight methodology to address NFRs early in agile software development processes. Activities such as elicitation, reasoning and validation are considered within the methodology in an effort to maintain agility while attempting to improve the quality of software developed with agile processes.

Given the contemporary state of reported evidence, we took a step forward to address the existing literature gap concerning security requirement verification in agile contexts. We proposed and evaluated an approach for reviewing security-related aspects in agile requirement specifications of web applications, previously presented in [57]. We decided to focus on web applications given that they have become a target for accessing or extracting sensible data. Results of our initial experimental evaluation, comparing our approach against using the complete list of OWASP high-level security requirements, indicated that using our approach has a positive impact on the performance of inspectors in terms of effectiveness and efficiency.

In this paper, we extend our previous study [57] in two ways: First, we provide further details on the approach and its initial evaluation, thus facilitating its adoption. Second, we conducted a new experimental trial which compares our approach with the perspective-based approach as reflected by Carver et al.'s contribution [12]. The results of this new

study strengthen our confidence in improving effectiveness and efficiency when using our approach.

The remainder of this work is organized as follows. Section 2 introduces the background of agile RE and how security verification is typically performed in this context. In this section, we also present related work. Section 3 presents in detail the technique we chose to evaluate our proposed approach, in this case, the perspective-based reading proposed by Carver et al. [12]. Section 4 introduces the approach we designed to deal with security verification in agile contexts. Section 5 presents the study design used to evaluate the approach. The results of the experimental trials are presented in Sect. 6. We discussed threats to validity and the results of the experiments in Sects. 7 and 8, respectively. Section 8.5 provides a discussion of limitations of our approach. Finally, our concluding remarks are presented in Sect. 9.

2 Background and related work

This section introduces the background on agile RE, inspections based on reading techniques, security properties and requirements and the synergy between these fields with verification activities. In addition, we describe related work to our proposed approach.

2.1 Agile requirements

The term "agile requirements" emerged in response to the agile manifesto. It is used to define the "agile way" of executing and reasoning about RE activities [31]. Yet, not much is known about the challenges posed by the collaboration-oriented agile way of dealing with RE activities. Ramesh et al. [45] performed a study with 16 organizations that develop software using agile methods. They identified that agile RE practices resulted in challenges regarding neglected NFRs, minimum documentation and no requirement verifications. The recent report from the *Naming the Pain in Requirements Engineering* initiative [24] (NaPiRE) extends already known challenges with (1) communication flaws between teams and customers, and (2) under-specified requirements that remain too abstract and, thus, are not measurable. These observations give a picture on the difficulties of dealing with NFRs in agile contexts. It is reasonable to believe that security requirements are no different in this respect.

2.2 Inspection based on reading techniques

Software inspection is a quality assurance method to detect defects early during the software development process. The aim is to guarantee that developers deal with complete, consistent, unambiguous and correct artifacts. In general, several authors have worked on quality assurance methods for verifying the quality properties of requirements. One of the most compared and evaluated methods, in several experiments and studies, is defect detection techniques, so-called reading techniques [28]. Software reading techniques attempt to increase the effectiveness of individual reviewers by providing guidelines that can be used to examine (by reading) a given software artifact and identify defects [53]. There is empirical evidence that software reading is a promising technique for increasing software quality for different situations and documents types [49].

Perspective-based inspection is a variant of a formal technical review. This type of inspection is based on explicitly defining the important stakeholders for a particular artifact and the types of issues that are of importance to the team. Rather than asking each reviewer to search for all types of problems, the perspective-based approach requires inspectors to examine the document using a role-based scenario based on how one specific stakeholder [12]. For example, an inspection of system requirements may include an inspector using a tester perspective. This inspector reviews the requirements by following a scenario in which he/she considers how to generate test cases based on the requirements.

Weak alignment of RE with inspection activities may lead to problems in delivering the required products in time with the right quality [7]; for instance, weak communication of requirement changes to testers may result in lack of verification of new requirements and incorrect verification of old invalid requirements, leading to software quality problems, wasted effort and delays [32].

2.3 Security properties and requirements

Security is an important quality characteristic of any software system that manages valuable data and processes [3, 13, 17]. Hence, software development should be conducted with security in mind at all stages and it should not be an afterthought [36]. However, developing secure software is not a trivial task often due to lack of awareness and security expertise in stakeholders and the inadequacy of methodologies to support developers who are not security experts [29]. Typically, security is often dealt with in retrospective and retrofitted when the system has already been designed and put into operation [38]. This causes defects that have a major impact on the project resulting in a higher cost to fix them.

Security properties are the targets the customer establishes for their security program. Without security properties, they do not know what they are trying to accomplish for security and therefore will not reach any goals [43]. Security requirement (SR) engineering can provide a foundation for developing secure systems. Nevertheless, like other quality requirements, they tend not to have simple yes/no satisfaction criteria. Haley et al. [27] present some challenges

related to SRs. First, people generally think about and express SRs in terms of "bad things" (negative properties) to be prevented. It is difficult, if not impossible, to measure negative properties. Second, for SRs, the tolerance on "satisfied enough" is small, often zero, given the implications of non-compliance. Moreover, stakeholders tend to want SR satisfaction to be very close to yes. Third, the effort stakeholders might be willing to dedicate to satisfying SRs also depends on the likelihood and impact of a failure to comply with them. In recent years, SRs has been investigated by several researchers. Mellado et al. [38] have conducted a systematic review of SR approaches to summarize existing methodologies. Fabian et al. [21] also provide a comparison of SR methods. Methods such as SQUARE [37] and Microsoft SDL [30] are compared.

2.4 Related work

Our related work was based on findings from literature reviews such as [1, 56] and empirical searches in indexed databases. For instance, Alsaqaf et al. [1] shows that very little is known about the evolution of NFRs (included security) in agile software development setting and more research is needed to understand the contexts in which these approaches would fit and add value, specially because very little empirical evaluation has been conducted. These studies revealed an important number of published proposals addressing security requirements in agile context. Most of them focused on analyzing, identifying and prioritizing these kind of requirements. On the other hand, we also found studies focused on showing challenges and practitioners perspectives in this context. However, few authors have addressed the specific problems of security verification activities ([12, 20, 43]). The picture is even poorer in the agile context as concluded in [56]. We are aware of only one study that explicitly states to address security verification activities in agile methods [18]. As far as security verifications are concerned, we include, as related work, some studies that do not explicitly mention their suitability in agile contexts, but given their domain application we consider them suitable to address the same direction of our approach.

Domah et al. [18] propose a lightweight methodology to address NFRs early in agile software development processes. NFR elicitation, reasoning and validation are considered within that methodology. Regarding verification, it depends on a quantification taxonomy with different levels of decomposition for identifying quantified validation criteria for each NFR. However, this methodology does not offer specific guidance to support inspectors in identifying security-related defects in requirement specifications. Hence, the previous knowledge on security is required to take advantage of the methodology.

Riaz et al. [46] describe a tool-assisted process for identifying key attributes of sentences to be used in security-related analysis and specification of functional security requirements using a set of context-specific templates. The tool takes natural language requirement artifacts (requirement specifications, feature requests, etc.) and a trained classifier for the current problem domain as input. It also parses the artifacts as text sentences and identifies which (if any) security properties relate to each sentence. The tool then presents the user with a list of applicable security requirement templates for the identified properties. We can say this work conceives security in the same way as our approach. The difference is that they propose an approach for identifying security requirements, whereas our approach focuses on detecting defects from security requirements already specified.

Elberzhager et al. [20] propose a model for security goals that involves guided checklists to support inspectors when checking security. They describe a step-by-step guide that results in questions to be checked by an inspector. This model is similar to our proposal because it works using a reading technique that supports the inspector on how to review security. However, there are differences. First, our approach focuses on verifying SRs in early stages, i.e., right after requirement specification and within agile requirement artifacts. Second, our approach addresses high-level SRs as defined by the Open Web Application Security Project (OWASP) [42], which provides a well-known industry standard on security. Furthermore, our proposal involves classifying the defects found by inspectors, providing a better understanding of the distribution of the problems.

Peine et al. [43] propose a model named Security Goal Indicator Tree (SGIT) that maps negative and non-local goals to positive, concrete features of the software that can be checked during an inspection. The model supports inspection of software documents from various phases of the development process. An SGIT links a security goal with numerous indicators (which may be beneficial or detrimental for the achievement of the goal) and structures the set of indicators by Boolean and conditional relationships enabling an efficient selection of indicator subsets. Despite the deep analysis provided by this work, it is not clear the level of expertise needed by inspectors to use the model. Furthermore, as the above related work, the model does not shed light on the type of defects detected.

Carver et al. [12] focus their proposal on a PBR technique with the aim of identifying security defects. They describe a set of perspectives that provide security-specific questions for a requirement inspection. Two of them are part of the PBR technique (designer and tester). They also created a new perspective based on the needs of a black hat tester. In this additional perspective, the reviewer focuses on three types of security information: cryptography, authentication

and data validation. According to the authors, those types of information and the related questions were adapted for requirements from Araujo and Curphey's article on security code reviews [2]. However, due to the large number of software vulnerabilities and the variety of ways to deploy computer attacks, it could not be enough to consider only three types of security controls. Indeed, the list is incomplete when compared to other security standards such as OWASP [42] or the common criteria [39].

To summarize, only few approaches exist that address the systematic detection of security defects, especially during early stages, and the number decreases when considering agile contexts which redraws the picture of how security could and should be dealt with [56]. However, we consider this last work, proposed by Carver et al. [12], the most related work given its nature to detect defects early via a reading technique and its focus on security. For that reason, we decided using this work to compare it against our approach with the aim of evaluating the suitability with respect to others proposals. In the following section, we present in detail the work proposed by Carver et al.

3 Perspective-based reading black hat approach

One of the objectives of Carver et al.'s work is to integrate practices from the security engineering and software engineering communities into a set of techniques for identifying and removing security vulnerabilities early in the software life cycle. This work is an adaptation of the PBR technique to address the security vulnerability problem during a requirement inspection process. The authors tailored PBR to focus on software security by augmenting two of the standard PBR perspectives (the designer and the tester) with additional security-specific questions. In addition, the authors proposed a new perspective based on the needs of a black hat tester. Using this approach, the inspector reviews the requirements by following a scenario in which he/she considers how to generate test cases based on the requirements.

3.1 Set of perspectives for requirement inspection

The **designer** perspective has the goal of ensuring that there is enough, consistent information present in the requirements to successfully create a system design. The existing scenario is augmented with questions that focus on whether important security-related information has been correctly specified rather than being left up to the designer, who may not be familiar with all details of the security policy. This perspective provides a set of questions that the reviewer should consider when following this perspective:

- Have the requirements specified enough information about the security policies for the designer to understand whether a layered security policy is required instead of a single point of vulnerability?
- If several administrator roles are defined, have they been defined as separate accounts with limited access to security resources or a single account with comprehensive super user permissions?

On the other hand, the scenario for the **tester** perspective remains unchanged, but is augmented with security-specific questions. The inspector using the tester perspective has the goal of ensuring that the trustworthiness of the system will be knowable during the testing phase. The questions of this perspective should consider the following:

- Have the requirements specified appropriate exception-handling functionality?
- Have the requirements specified adequate safeguards that would take effect once a malicious user has gained unauthorized access to the system?
- Does the system have a well-defined status, either a secure failure state or the start of a plausible recovery procedure, after a failure condition?

As a new perspective proposed by the authors, the **black hat** perspective focuses the reviewer on finding weaknesses in the requirements that could be exploited via an attack. The scenario that the reviewer follows is to create a set of malicious attack scenarios that seek to exploit system vulnerabilities.

3.2 Types of security properties

While creating the black hat scenario, the reviewer focuses on three types of security properties at the requirement stage: cryptography, authentication/authorization and data validation. These types of properties along with the related questions were adapted for requirements from Araujo and Curphey's article on security code reviews [2].

Cryptography relates to the encoding mechanisms specified for data items within the system. During the review, the inspector is looking for under-specified or incorrectly specified features that could be exploited. The questions include the following:

- Can the encoding mechanism specified for transmission and storage of data be broken?
- Do the cryptography mechanism specified follow well-known, well-documented and publicly scrutinized algorithms, and if not, can they be easily broken?

Authentication/authorization focuses the reviewer on determining how unauthorized users could gain access to the system. The questions include the following:

- Can the protocols for validating user identity be broken?
- If account lockout is specified, are there requirements in place to prevent denial-of-service attacks?
- Can user privileges be artificially elevated due to omission or poorly specified requirements?

Lastly, **data validation** is an important source of security vulnerabilities and focuses the reviewer on determining whether invalid data could be entered into the system. The question is: Do the requirements leave any opportunities for invalid data to be entered by the lack of validation of external data?

4 Our approach

In this section, we present our approach for reviewing security-related aspects in agile requirements specifications of web applications. The approach was designed considering user stories and their security specifications as input and involves applying natural language processing (NLP). The goal is to relate those user stories to candidate security properties and high-level security requirements proposed by the Open Web Application Security Project (OWASP), a well-known online community that produces freely available articles, methodologies, documentation, tools and technologies in the field of web application security [42]. As a result, the approach provides a user story focused reading technique that can be used to support the manual inspection of agile requirements. The reading technique helps to verify the user story security specifications against the OWASP high-level security requirements to identify defects such as omissions, ambiguities, inconsistencies or incorrect facts. In the following, we provide more details of the conception of the approach.

4.1 Assumptions

The approach was designed with some underlying assumptions in mind. These assumptions are as follows.

Requirements are specified in a user story format The software industry has gradually increased the use of agile and hybrid methods in its projects [33]. In this context, user story is the most frequently used artifact for requirement specification [48]. Therefore, the approach is focused on agile and, more specifically, on user stories. The stories are often analyzed independently and structured in a sentence as follows: As a [*role*], I want to [*feature*], so that [*reason*].

The OWASP represents a reliable baseline and standard of security guidelines OWASP has a strong focus on web applications, one of the targets of our approach. OWASP concerns providing practical information about security in web applications to individuals, corporations, universities, government agencies and other organizations worldwide. Many open source security-related tools (e.g., SonarQube[1]) and current research (e.g., [51]) on web application security use OWASP as a definitive reference. Hence, we consider the reliability of this project as a reasonable assumption.

4.2 Approach scope delimitation

Hereafter, we answer some potential questions to provide further understanding of the intended approach.

To whom is the approach intended? Our approach was designed to support novice inspectors and junior security analysts. This work provides them with a reading technique to assist in the identification of defects related to security aspects in agile requirements specifications. According to Nerur [40], people with a high-level of competence are of vital importance in agile teams because they tend not to depend on documentation to fulfill their functions. Much of the knowledge in agile development is tacit and resides in the heads of the development team members [8]. Even more challenging, competence in software security is typically not widely spread among agile practitioners [26]. Given this, it is helpful to guide novice inspectors, including junior security analysts, by providing a detailed reading technique to support them. We believe that senior security analysts could still use the approach, but they are outside of the scope of our evaluation.

What security-related aspects does the approach cover? We decided to focus on security properties and high-level SRs as proposed by the OWASP [42]. These high-level SRs describe the most important security features that architects and developers should include in every web application [42]. The System and Software Quality Requirements and Evaluation (SQuaRE) model [59] also define security characteristics, which hereafter, for term compatibility, will also be referred to as security properties. OWASP contains three security properties: confidentiality, integrity and availability. SQuaRE, in contrast, contains five: confidentiality, accountability, integrity, non-repudiation and authentication. Based on their definitions, all of the SQuaRE security properties can be mapped onto the OWASP security properties. For our final list of considered security properties, we used the OWASP properties with a single change, splitting confidentiality into two separate properties: (1) confidentiality and

[1] https://docs.sonarqube.org/latest/user-guide/security-rules/.

Table 1 Security properties considered by our approach

Security property	Description
Confidentiality (C)	Degree to which the data are disclosed only as intended
Integrity (I)	Degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data
Availability (A)	Degree to which a system or component is operational when required for use
Identification authorization (IA)	Degree to which the identity of a subject or resource can be proved to be the one claimed

Table 2 Defect types' definition and examples in scope of our approach

Defect type	Definition	Applied to security
Omission (O)	Necessary information about the system has been omitted from the software artifact	One or more security requirements that are not covered by the specifications originally created by an agile team
Ambiguity (A)	A requirement has multiple interpretations due to multiple terms for the same characteristic	For example, "the system shall inactivate a session when it exceeds certain periods of inactivity" is ambiguous because it is not clear the amount of time necessary to inactivate the session It could be seconds, minutes or hours
Inconsistency (IS)	Two or more requirements are in conflict	One security requirement specifies to encrypt data with RSA algorithm, but another one specifies to encrypt it with AES
Incorrect Fact (IF)	A requirement asserts a fact that cannot be true under the conditions specified for the system	For example, "the system shall protect the firewall" does not make sense because it is the firewall that protects the system

(2) identification and authentication. Table 1 presents the security properties considered by our approach.

What types of requirements defects does the approach cover? In RE, a defect can be defined as any problem of correctness and completeness with respect to the requirements, internal consistency or other quality attributes [53]. A common defect taxonomy used when inspecting requirements is the one proposed by Shull [49]. The defect types in this taxonomy are: omission, ambiguity, inconsistent information, incorrect fact and extraneous information. However, we excluded the extraneous information defect type (which concerns specifying requirements that are not needed). This decision was taken because we use the OWASP high-level SRs as a reference; while they are stated as mandatory for inclusion, they are not necessarily complete, given that specific security needs may sprout for specific applications. Hence, given the impact that a missing security requirement can have on the application, we did not feel comfortable to recommend exclusions. Table 2 shows the defect types covered by our approach, their definitions and examples applied to security.

What kind of review technique does the approach use? Typically, developers and software analysts rely on ad hoc methods or checklists to analyze documents. In an ad hoc review, the reader is not given directions on how to read. The result is that reviewers tend to build up skills in document understanding slowly based on individual experiences acquired over time [4]. For this reason, we decided to focus the review of our approach on a reading technique to increase the effectiveness of individual reviewers by providing a systematic guide that can be used to examine, in our case, security-related aspects and consequently to identify defects.

To which part of the life cycle of agile methods can the approach be applied? Agile methods are characterized by having iterative structures that should allow early delivery, continual improvement and rapid and flexible response to change [6]. Hence, we envision that our approach is used just before a user story is defined as ready for codifying. In this way, we avoid rework that can be caused by not considering a security control or integrating one requirement with another one.

4.3 Overview of our approach

We propose our approach in two defined phases: (1) generating artifacts for the reading technique based on the agile requirements specifications and (2) following the reading technique to identify defects. Figure 1 shows the flow and relationships between the artifacts and phases that form our approach.

4.3.1 Phase 1: generating artifacts for the reading technique

To generate the artifacts to follow our reading technique, we use natural language processing (NLP) to extract keywords from the user story. Thereafter, these words are used

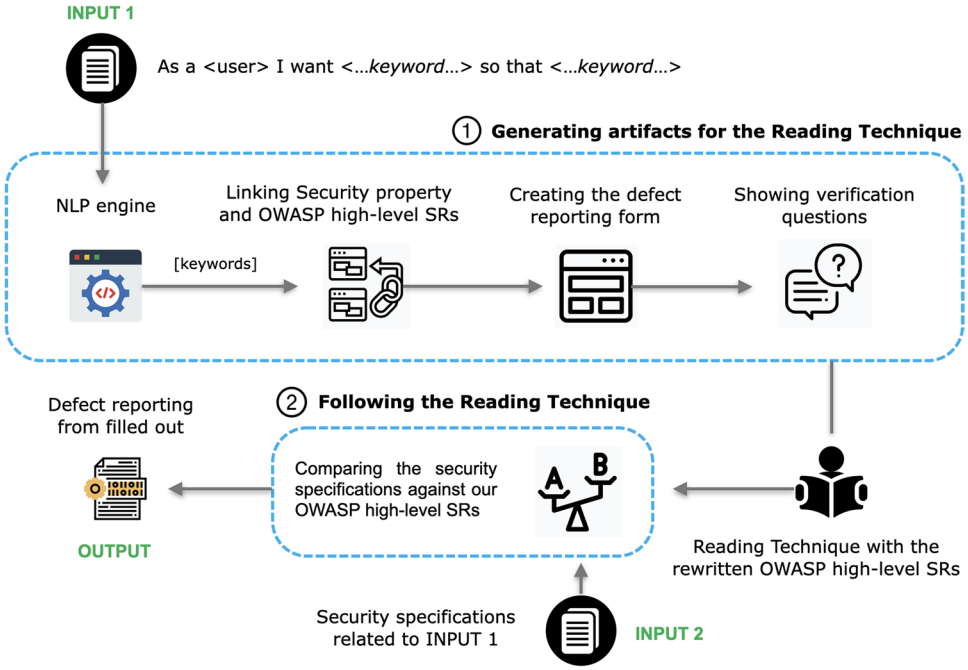

Fig. 1 Overall structure of our approach

Table 3 Way to extract the keywords from the user story

Type of word	User story skeleton
Verbs	As a [user], I **[want to]**, [so that]
Nouns	As a [user], I [want to], **[so that]**

to identify security properties and to link the related OWASP high-level SRs to be verified. The availability of automatic tools for the quality analysis of natural language requirements is recognized as a key factor for achieving software quality [34]. Details on how keywords and security properties are identified follow.

Extracting keywords This activity involves automatically analyzing a user story that describes the features and functional requirements of the software to be built. Our approach extracts the relevant verb (action) of the user story that indicates a potential behavior to consider when thinking about security. For instance, the verb "export" could indicate confidentiality concerns because it is an action about transporting something. On the other hand, the verb "modify" could indicate integrity concerns because it is an action that involves altering something. In some cases, the nouns of the user story can also indicate situations where certain security features should be considered. This is particularly important to identify availability needs, e.g., time values (day, hour, second, period) may indicate scale/performance restrictions of the software. In that sense, nouns are also extracted for matching purposes. In summary, the verb is extracted from the second block of the user story format and nouns are extracted from the third block. Table 3 shows where the words come from.

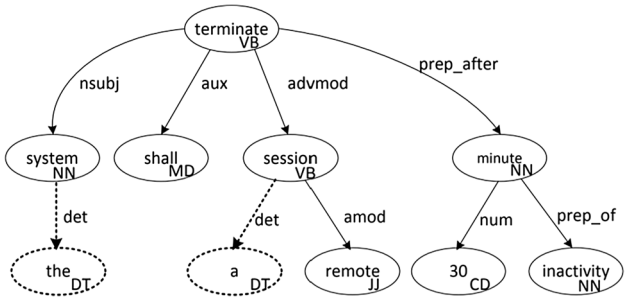

Fig. 2 Sentence representation [50]

To extract the words, we developed a software framework (FESRAS),[2] which uses the Stanford CoreNLP tool[3] through a library that provides a set of natural language analysis tools written in Java. The library represents each sentence as a directed graph where the vertices are words and the edges are the relationships between them. Therefore, the software framework can take the verbs and nouns of the user story. Figure 2 shows how the Stanford CoreNLP tool represents the user story to identify verbs, nouns, among other kind

[2] https://github.com/hrguarinv/FESRAS.

[3] https://github.com/stanfordnlp/CoreNLP.

Table 4 Relationship between the keywords and security properties

Keyword	Confidentiality	Integrity	Availability	IA
Access	X			X
Alter		X		
Apply				X
Auto-populate		X		
Change		X		
Create		X		
Define				X
Delete		X		
Display	X			
Establish				X
Export	X			
Generate		X		
Modify		X		
Read	X			X
Recover			X	
Backup			X	
Day			X	
Hour			X	
Password				X
Period			X	
Privilege				X
Role				X
Time			X	

of words. For this, consider the sentence: The system shall terminate a remote session after 30 minutes of inactivity.

Note that the words extracted by using the Stanford Core NLP are not always keywords. After extracting the verb and nouns, we need to determine whether these words match any keyword of our repository. If so, the user story contains at least one keyword that indicates some security concern that should be addressed. This matching is explained below.

Identifying Security Properties and Linking High-Level SRs After identifying the keywords of the user story, we need to identify security properties in order to map high-level SRs that represent a set of security-specific features to be verified. We use the keywords extracted from the user story to map security properties. Our software framework contains a set of keywords that are related to at least one security. Table 4 shows some these keywords that are part of our repository, to indicate which security properties should be considered. As an example, observe that the keyword "access," which is a verb, indicates security concerns related to confidentiality, identification and authorization, because when accessing data, controls must be in place to guarantee the correct disclosure to it. Our online material, available at Zenodo,[4] contains all the keywords of the repository.

This repository is based on a similar one provided by Slankas and Williams [50] in their work about automated extraction of NFRs in available documentation. However, we complement it by (1) considering additional keywords stated by the OWASP and (2) including synonyms of the words from these sources. By doing this, we increase the coverage of our repository. If there is no match between the words extracted and the keywords of our repository that indicates security properties, our approach will link the user story with their security specifications to all the security properties stored in our repository. With this, the inspection will not be as advantageous in terms of effort and time as we anticipated because the reviewer will have to evaluate each of the security properties. However, in this way we ensure

Table 5 OWASP high-level security requirements by security property

Security property	OWASP high-level security requirements
Confidentiality	C1. Data shall be protected from unauthorized observation and disclosure both in transit and when stored
	C2. System sessions shall be unique to each individual and cannot be shared
	C3. System sessions are invalidated when timed out during periods of inactivity
	C4. TLS protocol shall be used where sensitive data are transmitted
	C5. System shall use strong encryption algorithm at all times
Integrity	I1. Any unauthorized modification of data must yield an auditable security-related event
	I2. All input is validated to be correct and fit for the intended purpose
	I3. Data from an external entity shall always be validated
Availability	A1. The application server shall be suitably hardened from a default configuration
	A2. HTTP responses contain a safe character set in the content type header
	A3. Backups must be implemented and recovery strategies must be considered
Identification and authorization	IA1. Users are associated with a well-defined set of roles and privileges
	IA2. The digital identity of the sender of a communication must be verified
	IA3. Only those authorized are able to authenticate and credentials are transported and stored in a secure manner
	IA4. Passwords treatment must include complex passphrases, options to recover and reset the password and default passwords not allowed

[4] https://doi.org/10.5281/zenodo.3966542.

Fig. 3 Defect reporting form

User Story	Security Property	OWASP High-level Security Requirement	Omission	Ambiguity	Inconsistency	Incorrect fact
1	Integrity	Security requirement a				
		Security requirement b				
		Security requirement c				
2	Availability	Security requirement d				
		Security requirement e				
		Security requirement f				

Table 6 Verification questions for the different defect types

Type of defect	Question
Omission	When comparing the security specifications with the OWASP high-level SRs, are there high-level SRs or characteristics that were not specified?
Ambiguity	Does any security specification allow for multiple interpretations?
Inconsistency	Are there two or more security specifications in conflict?
Incorrect fact	Is there any security specification stating information that is not true under the conditions specified?

that the user story is examined by the four main domains of security in web applications.

After identifying the security property, we need to link the high-level SRs, which according to OWASP are basic to deal with security in web applications. Table 5 shows the OWASP high-level SRs by security property.

At this point of our approach, we already identified the keywords of the user story, the security properties related to those keywords and consequently the OWASP high-level SRs that address those security properties. The next step is to build the defect reporting form that works as a model and synthesizes most of the information the inspector needs to identify defects in the specifications. This form presents, in a structured way, much of the information that the reviewer must analyze to identify and classify the defects. Information such as the user story, the security property, the OWASP high-level SRs and the defect types are provided by the form. Figure 3 shows a template of this form with two of the security properties covered by our approach.

With the defect reporting form ready to be used, the inspectors can use it as a model to verify whether the security specifications built, in a realistic example, by requirements or security analysts, contain any of the defect types covered in this work. This happens when inspectors compare the security specifications against the OWASP high-level SRs presented in the reporting form. To reach this, our approach formulates a set of verification questions to help identifying the different defect types in the security specifications. Table 6 shows the questions.

The first question aims to identify omission defects. For this type of defect, we provide to inspectors our rewritten OWASP high-level SRs that work as a model. With this, we seek to identify which SRs, stated by OWASP as basic and essential, are missing in the security requirements that were specified in agile software projects. Note that in the worst case, few or no security requirements are specified, as typically occurs in this type of projects. Thus, by doing this comparison we can offer relevant insights to inspectors to identify this type of defect. To identify the remaining defects, we use their definitions in a clear and short way. Our intention is to provide agile support to increase the efficiency of the inspection task. We believe that inspectors can directly associate the concept of the defect with the actions that allow them to identify such defects. For instance, to detect ambiguity defects our reading technique asks the inspector if any security specification allows for multiple interpretations. Regarding inconsistency defects, the inspectors must focus on figuring out whether two or more security specifications are in conflict. In this way, the meaning of the defect seeks to guide the inspector on the detection of ambiguity, inconsistency and incorrect fact defects.

4.3.2 Phase 2: following the reading technique

This phase aims to guide the reviewer in finding the requirement defects. Using the artifacts generated in Phase 1, we propose a reading technique that inspectors can follow to answer the verification questions in order to look for defects. As presented in Sect. 2.2, software reading techniques help a reviewer to "read" a requirement artifact for the purpose of finding relevant information that gives specific and practical guidance for identifying, in this case, security-related defects in agile requirements specifications.

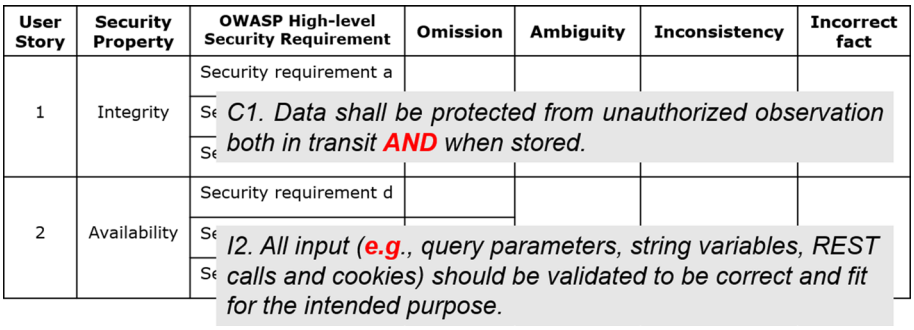

Fig. 4 Visual aspect of the rewritten OWASP high-level SRs

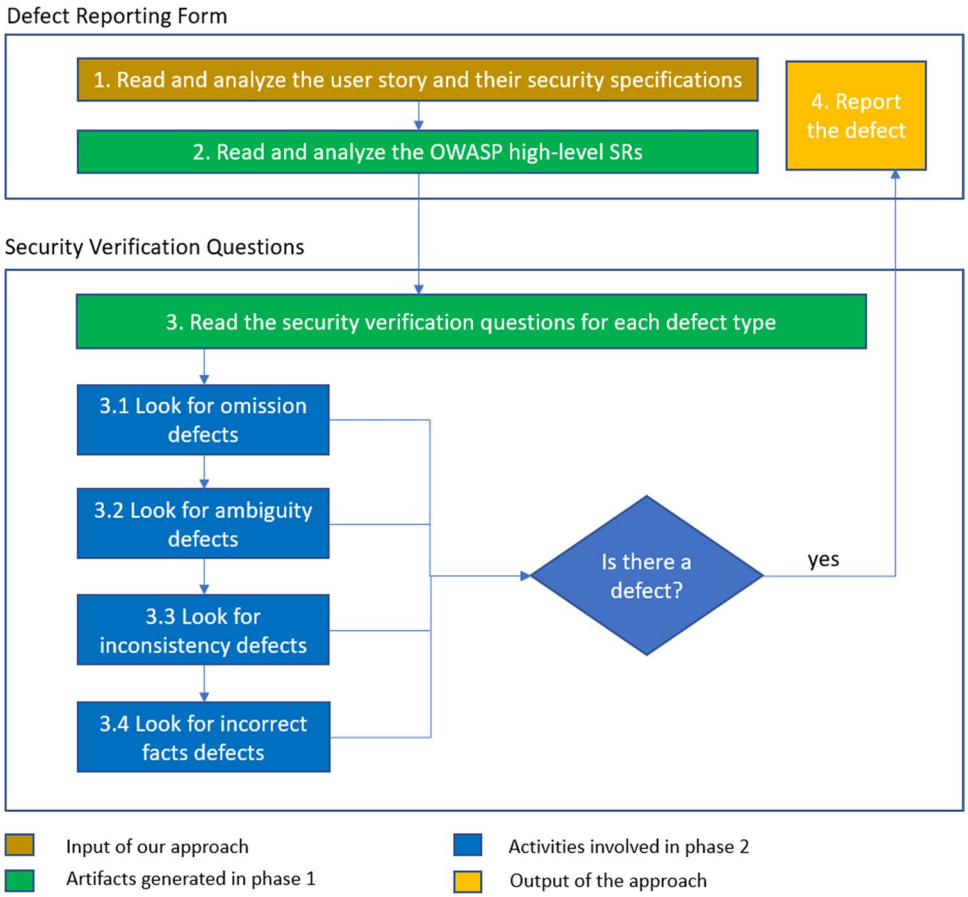

Fig. 5 Procedures to apply to each user story

To facilitate the review, our approach rewrites the OWASP high-level SRs in such a way that inspectors can easily identify certain security aspects. For instance, we use the "AND" logical connector in capital letters to get the attention of the reader and indicate that both aspects must be considered to satisfy the high-level SR, e.g., we have the following to the first confidentiality high-level SR: *C1. Data shall be protected from unauthorized observation or disclosure both in transit **AND** when stored*. In this case, if the specifications were well specified, they must consider security aspects related to data protection both in transit and in storage. Otherwise, there is an omission defect.

On the other hand, the security statements provided by our reading technique also present examples for some concepts in order to give inspectors an idea about the context of the OWASP high-level SRs. An example to the second integrity high-level SR follows: *I2. All input, **e.g.**, query parameters, string variables and cookies, is validated to be correct and fit for the intended purpose*. In this way, reviewers are provided with a reading technique that should increase their performance during the review. Figure 4 shows an example of how the OWASP high-level SRs looks to get the attention of inspectors.

Table 7 Input of the approach as agile requirements specification

User story	Security specification (SS)
As a customer, I want to be able to export my personal information so that I can use it in other systems	SS1. The system shall ensure that there is no residual data exposed SS2. The system shall store credentials securely using the AES encryption algorithm SS3. The system shall use the RSA encryption algorithm to protect all data all the time SS4. The system shall deactivate a session when it exceeds certain periods of inactivity SS5. The system shall encrypt the roles and privileges of the system

Fig. 6 Matching keywords to security properties

Keyword	Security Properties			
	Confidentiality	Integrity	Availability	Identification & Authorization
Access	X			X
Apply				X
Auto-populate		X		
Change		X		
Create		X		
Define				X
Delete		X		
Display	X			X
Establish				X
Export	**X**			
Generate		X		
Grant				X
Log in/out				X
Modify		X		

To summarize how inspectors should use our approach, we present Fig. 5 that provides step-by-step activities to be followed by inspectors. The process begins by reading and analyzing the user story and their security specifications (input of our approach) to compare them with the OWASP high-level SRs. After this, inspectors can read our security verification questions for each defect type with the aim of detecting them and then report them (output of our approach). Please note that the process is repeated for each user story with their respective security specifications. We clarify that if there are no related security specifications as input in our approach, the result will be all of the OWASP high-level SRs related to the security property matched by the words extracted from the user story.

4.4 Motivational example

In the following, we demonstrate the application of our approach in an example setting. For this purpose, we present one of the agile specifications used in the experiments. Table 7 shows a user story and its set of security specifications (inputs of our approach) with some defects commonly applied to a web application.

With the user story in sight, the software framework extracts the words of the second and third blocks of the user story by using the Stanford CoreNLP and then evaluates whether there is match to link the security properties. In this case, the extracted words are "export" (from the second block) and "system" (from third block). The following illustrates which keywords are extracted from the user story.

> *As a customer, I want to be able to **export** my personal information so that I can use it in other **systems**.*

Thereafter, the framework can verify whether some security property is related to the extracted words. According to Table 4, "export" matches confidentiality, while "system" does not match any of the security properties. Figure 6 presents the relationship between the keyword and the security property.

In this way, since our approach identified confidentiality as security property, it can propose OWASP high-level SRs (*Cf.* Table 5, confidentiality). As part of our approach, we

Table 8 Defects reporting form

User story	Security property	OWASP high-level SRs	O	A	IS	IF
US1	Confidentiality	C1. Data shall be protected from unauthorized observation AND disclosure both in transit AND when stored.		SS4	SS2	SS5
		C2. System sessions shall be unique to each individual AND cannot be shared.	X		SS3	
		C3. System sessions are invalidated when timed out during periods of inactivity.				
		C4. TLS protocol shall be used where sensitive data are transmitted.	X			
		C5. System shall use strong algorithms (e.g., DES, AES, RSA) to encrypt data				

rewrite those OWASP high-level SRs to improve its readability and understanding. For that reason, the OWASP high-level SRs are provided to inspectors in the following way.

- C1. Data shall be protected from unauthorized observation and disclosure both in transit AND when stored.
- C2. System sessions shall be unique to each individual AND cannot be shared.
- C3. System sessions are invalidated when timed out during periods of inactivity.
- C4. TLS protocol shall be used where sensitive data are transmitted.
- C5. System shall use strong algorithms (e.g., DES, AES, RSA) to encrypt data.

These OWASP high-level SRs are the basis to determine whether the security specifications presented in Table 7 contain omission defects. Note that inspectors are encouraged to read our verification questions to further analyze the quality of the specifications.

Our approach then generates the defect reporting form by showing the user story with the linked security properties and OWASP high-level SRs. The verification questions are also provided here. Thus, inspectors know which security aspects they should verify. In that sense, the verification process starts at this point. By having inspectors responding to the verification questions looking for defects, we expect to obtain valuable insights from them on the quality of the security specifications. A sample enactment of answering these questions follows.

When comparing the security specifications with the OWASP high-level SRs, are there high-level SRs or characteristics that were not specified? In this case, 3 out of 5 confidentiality high-level security requirements linked in Table 5 are related or make sense to the security specifications. However, we have two unspecified high-level requirements. Note that the second confidentiality high-level SR (C2) and the fourth confidentiality high-level SR (C4) are not covered by the security specifications. Therefore, we have detected two defects that should be marked as "omission."

Does any security specification allow for multiple interpretations? If we analyze with caution, we see that the fourth security specification (SS4) reflects a weak statement as the amount of time concerning "certain periods of inactivity." It could be hours or seconds. Thus, we have identified a defect related to ambiguity.

Are there two or more security specifications in conflict? The answer is affirmative. The second security specification (SS2) and third security specification (SS3) conflict because SS3 indicates to encrypt all data using the RSA algorithm. Nevertheless, SS2 indicates to protect credentials, which are also data, using the AES algorithm. Thus, we have identified a defect related to inconsistency.

Is there any security specification stating a characteristic that cannot be true under the conditions specified for the system? The fifth security specification (SS5) is not correct because the concepts of the system cannot be encrypted. The action "encrypt" is not correct in the statement.

Finally, the reviewers fill out the defect reporting form that summarizes the defects found. Table 8 presents the output of the review using the reading technique. Note that the O column is related to the OWASP high-level SRs that were omitted to satisfy the security property (SP). The other columns are related to the remaining defect types.

In summary, this table indicates that the security specifications related to the user story contain six defects. Two out of them were marked as omission because the second and fourth OWASP high-level SRs related to confidentiality (C2, C4) are not covered by the security specifications. The rest of the defects (4) are related to ambiguous, inconsistent and incorrect fact defects. In this case, the fourth security specification (SS4) was marked as ambiguous, the second security specification (SS2) and the third security specification (SS3) were marked as inconsistent and the fifth security specification (SS5) was marked as incorrect.

5 Experiment

We evaluated the approach by conducting three controlled experimental trials with eight graduate and 48 undergraduate computer science students of the Pontifical Catholic University of Rio de Janeiro. The focus was to observe the impact

Table 9 Controlled experimental trials conducted across the study

Study	Trial	Date	Undergraduate	Graduate	Total
Original	1	March 2019	25	0	25
	2	March 2019	0	8	8
New	3	November 2019	23	0	23

of using our approach on effectiveness, efficiency, usefulness and ease of use.

We evaluated the study in two phases. In the first, we wanted to know whether our approach was suitable under certain non-complex conditions. With some positive results, the second phase was planned. In this case, the goal was to obtain more empirical evidence and then determine whether our approach is technically promising to use it in more complex environments. Thus, we conducted a new study by analyzing the effectiveness and efficiency of our approach when compared with the PBR Black Hat approach proposed by Carver et al. [12]. This decision was supported due to that approach uses the same type of inspection technique that our approach and therefore is totally aimed at identifying security-related defects.

The motivation for conducting the new study is to examine whether the findings can be replicated in different levels of support (e.g., using the PBR Black Hat approach), incorporating lessons learned from the first evaluation. We considered guidelines for reporting additional experimental studies proposed in [11].

In this section, we detail the design of the experiments conducted to evaluate our approach. We present the goal, hypotheses, variable selection, selection of subjects, instrumentation, among others. We break down the experiments in three trials with different characteristics. In the original study, we allocated undergraduate and graduate students of Computer Science that were divided into two trial experiments. Regarding the new study (third trial), we assigned undergraduate students. Table 9 presents some relevant details of the trials conducted across the study.

In the following, we present similarities and differences between the original and the new study. In all the studies, subjects were assigned to the task of identifying security-related defects based on a set of agile requirements specifications.

5.1 Goal, hypotheses and research questions

For all the experiments (original and new study), we wanted to determine whether the use of our approach leads to efficient and effective detection of security-related defects when compared to an ad hoc inspection based on personal expertise and the PBR Black Hat approach. Table 10 details the definition of this study's goal by following the template provided by Basili [5].

Based on our goal, we formulated research questions (RQs) that seek to address two aspects that we believe should be covered by our approach. First, efficiency and effectiveness play an important role in inspection activities, as inspectors should be able to find defects (effectiveness) with reasonable effort (efficiency). Second, we also address usefulness concerns, because they are closely related to the adoption of the approach. Therefore, we defined the following two RQs that apply to both the studies conducted.

- **(RQ1)** Does our approach have an effect on defect detection effectiveness and efficiency when compared to the other review approaches of the study?
- **(RQ2)** How do the inspectors perceive the usefulness and ease of use of our approach?

Based on RQ1, we derived hypotheses to be evaluated quantitatively. Note that we defined hypotheses that vary depending on the study performed and that they may refuted or supported only in comparison with the considered other methods. In the original study, the following hypotheses were derived:

- $\mathbf{H}_{01a}$ There is no difference in terms of effectiveness when using our approach, when compared to using the OWASP high-level SRs.
- $\mathbf{H}_{11a}$ There is a difference in terms of effectiveness when using our approach, when compared to using the OWASP high-level SRs.

Table 10 Goal definition of the experiments

Analyze	The reading technique generated by our approach
for the purpose of	Characterization
with respect to	The effectiveness, efficiency, usefulness and ease of use of the approach
from the point of view of	Researchers (on the measured effectiveness and efficiency) and inspectors (on the perceived usefulness and ease of use)
in the context of	Novice inspectors using our approach, when compared to using the OWASP high-level SRs and the PBR Black Hat approach

Table 11 Metrics used to answer the RQs and test the hypotheses

Criteria	Type	Description
Effectiveness	Quantitative	Ratio between the number of defects found and the total of seeded defects in the specifications
Efficiency	Quantitative	Ratio between the number of real defects found and the time spent in finding them
Usefulness	Quantitative	Frequency of the participants' perception on the usefulness of the approach using a follow-up questionnaire
	Qualitative	Coding of the answers of the follow-up questionnaire
Ease of use	Quantitative	Frequency of the participants' perception on the ease of use of the approach using a follow-up questionnaire
	Qualitative	Coding of the answers of the follow-up questionnaire

- $\mathbf{H}_{02a}$ There is no difference in terms of efficiency when using our approach, when compared to using the OWASP high-level SRs.
- $\mathbf{H}_{12a}$ There is a difference in terms of efficiency when using our approach, when compared to using the OWASP high-level SRs.

Regarding the new study, we compare the performance of using our approach against the PBR Black Hat approach. Thus, we defined the following hypotheses.

- $\mathbf{H}_{01b}$ There is no difference in terms of effectiveness when using our approach, when compared to using the PBR Black Hat approach.
- $\mathbf{H}_{11b}$ There is a difference in terms of effectiveness when using our approach, when compared to using the PBR Black Hat approach.
- $\mathbf{H}_{02b}$ There is no difference in terms of efficiency when using our approach, when compared to using the PBR Black Hat approach.
- $\mathbf{H}_{12b}$ There is a difference in terms of efficiency when using our approach, when compared to using the PBR Black Hat approach.

5.2 Variable selection

The independent variable is the treatment applied by the groups to detect defects in the SR specifications. In that sense, inspectors who were part of the control group in the first and second trial received as support the OWASP high-level SRs. On the other hand, in the third trial, inspectors who were part of the control group received the PBR Black Hat approach, but at the same time, they received the same support that inspectors in the first and second trials; that is, they also received the OWASP high-level security requirements. With respect to the experimental group, all the inspectors received our approach.

Regarding the dependent variables, we used effectiveness and efficiency, defined as follows. *Effectiveness* is expressed as the ratio between the number of real defects found and the total of seeded defects in the documents. On the other hand, *Efficiency* refers to the ratio between the number of real defects found and the time spent in finding them. For these variables, we collected quantitative data to test the hypotheses presented in Sect. 5.1. We also collected qualitative data with open questions via a follow-up questionnaire. The aim was to gain insights about the perceived usefulness and ease of use of the approach. Table 11 summarizes the metrics collected in the experiments.

5.3 Selection of subjects

Our subjects were intended to represent novice inspectors. We thus selected subjects, by convenience, from classes on computer science at PUC-Rio, involving, for the original study, 25 undergraduate (first trial) and eight graduate students (second trial). Regarding the new study, the experiment was conducted in one trial (third trial), involving 23 undergraduate students from classes on computer science at PUC-Rio. There is evidence that using students is an effective way to advance software engineering theories and technologies, but, like any other aspect of study settings, should be carefully considered during the design, execution, interpretation and reporting of an experiment [22].

We characterized the subjects by their experience and knowledge in five areas: agile software development (ASD), agile RE (ARE), security aspects (SA), security requirements (SRs) and requirement inspections (RI). To this end, we defined a scale from one to five where lower score indicates lower experience and high score indicates experience in the industry.

Table 12 shows details of the characterization of the students of the original study. Subjects with at least three values greater to three were highlighted to identify the participants were equally divided between the groups of the experiments, applying the blocking principle. We did not list subjects of the new study because none of them met the requirements to be treated differently. Therefore, they were randomly assigned into control and experimental groups.

As a result, we found that the majority of students had a low level of security and requirement inspection experience. Hence, they match our intended profile (novice inspectors).

Table 12 Details of the characterization of the subjects

Experiment	Level	ID	Trial	ASD	ARE	SA	SR	RI
Original	Undergraduate	1	1	3	3	1	1	2
		2	1	2	1	1	1	2
		3	**1**	**4**	**4**	**4**	**4**	**2**
		4	**1**	**4**	**1**	**4**	**3**	**4**
		5	**1**	**5**	**2**	**4**	**4**	**2**
		6	1	2	2	2	1	2
		7	1	2	2	3	3	3
		8	1	2	2	2	1	1
		9	1	3	2	1	1	2
		10	1	5	2	2	1	2
		11	1	2	1	2	2	1
		12	1	3	2	1	2	2
		13	1	1	1	1	1	1
		14	1	2	5	3	3	1
		15	1	2	2	2	2	2
		16	1	5	5	2	1	3
		17	1	3	1	1	1	1
		18	1	3	1	3	2	2
		19	1	4	1	1	2	2
		20	1	5	5	2	2	1
		21	1	4	4	1	1	1
		22	**1**	**5**	**5**	**2**	**2**	**4**
		23	1	2	3	2	2	2
		24	1	4	4	1	1	1
		25	1	2	2	3	2	1
	Graduate	26	2	3	2	3	3	1
		27	2	2	2	3	3	1
		28	**2**	**4**	**2**	**4**	**4**	**2**
		29	2	4	2	1	1	1
		30	2	4	1	2	2	2
		31	**2**	**4**	**3**	**4**	**4**	**5**
		32	2	2	2	2	3	3
		33	2	2	2	2	2	2

5.4 Experimental context

In the following, we detail the experimental context of the studies. Before conducting the controlled experiment of the first trial, we carried out a pilot study with two independent volunteers. The aim was to evaluate the overall (particularly technical) feasibility, time and adverse events and improve the experimental materials before the experimental trials.

All the studies were conducted with the same agile specifications. These requirements contain a set of user stories in this format: As a [*Role*], I want [*Feature*], so that [*Reason*]. The document also contained their related security specifications with seeded defects that represent specifications that in real settings would be created by requirement analysts or product owners in agile teams (cf. Sect. 4.4 to see one of the specifications used in the experiments).

Aiming at mitigating threats to validity concerning the distribution of subjects between groups, we characterized the subjects and then applied the principles of balancing, blocking and random assignment [58]. In all the trials, students who demonstrated experience on the topics involved in the study were separated and distributed equally into the control and experimental group. In the case of the first and second trials, we allocated equally 6 out of 33 between the

Table 13 Support provided to each group of the experiments

Trial	Control group	Experimental group
1, 2	OWASP high-level SRs	Our approach
3	PBR Black Hat + OWASP high-level SRs	Our approach

Table 14 Number of subjects by experiment, trial and group

Experiment	Trial	Control	Experimental	Total
Original	1	12 US	13 US	25
	2	4 GS	4GS	8
New	3	11 US	12 US	23

Table 15 Distribution of the seeded defects

User Story	Omission	Ambiguity	Inconsistency	Incorrect fact	Total
US1	2	2	2	1	7
US2	2	2	2	1	7

Table 16 Context factors of the experiments

Context factors	Original study	New study
Subjects	25 US and 8 GS	23 US
Setting	In-class activity	In-class activity
Training provided	–Security properties and OWASP	–Security properties and OWASP
	–Type of defects	–Type of defects
		–Our reading technique and PBR Black Hat approach
Other changes	NA	–Reminder to follow the task description

groups, given they had higher scores. Regarding the new study, subjects were randomly placed into the experimental and control group since the characterization did not shed light. Table 13 shows the support given to the control and experimental group in each trial.

In Table 14, we present the number of undergraduate students (US) and graduate students (GS) in each group of the trials.

Subjects who found less than two defects were discarded as outliers because, in our understanding, their results reflect a lack of interest in having a good performance in the review. In the original study, we discarded two subjects from the first trial and one for the second trial. All these discarded subjects conducted the inspection by using the OWASP high-level SRs (control group). Regarding the new study, we discarded one subject from the third trial. In this case, the discarded subject conducted the inspection by using our approach (experimental group).

To avoid the defect seeding to represent a confounding factor, the type and amount of seeded defects to evaluate the suitability of our approach was carefully considered. All the trials were conducted with the same type and distribution of defects. Table 15 shows the distribution of the seeded defects per user story. In total, 14 defects were seeded. Three independent researchers reviewed the representativeness of the requirement specifications and the defects before conducting the experimental trials.

The original (first and second trials) and the new study (third trial) differed in terms of certain aspects related to the setting of the study. In the original study, 33 subjects participated; they were divided into two trials with 25 undergraduate and eight graduate students. In the new study, we involved 23 undergraduate students. This number of participants was defined according to the availability of students to whom we had access and who met our study profile. We conducted the experiments as an in-class activity to have the attendance of most of our subjects. We motivated the subjects to give their best, while we guaranteed the confidentiality of their results. Regarding training, in the original study we did not provide training on inspection techniques because, at that time, we considered that our approach and our task description could be self-assimilated by the students. After the lessons learned from the first study, we identified that training should be provided in the new study. For that reason, we provided training on both, our approach and the PBR Black Hat approach. Additionally, we reminded inspectors to pay attention to the task description according to the treatment provided. These changes were motivated by the feedback received by the inspectors in the follow-up questionnaire of the first two trials. Table 16 presents the main factors involved in the experiments to summarize the context of the original and the new study.

5.5 Experimental design

Our experiments were composed of one factor with two treatments: (1) using our proposed reading technique and (2) using the OWASP high-level SRs (first two trials) or using the PBR Black Hat approach (third trial). Figure 7 shows all the phases involved in the experiments.

The study design is composed of a set of artifacts distributed into three phases. Details of the artifacts used in the experiment are available at Zenodo.[5] In the first phase, all the students filled out a characterization questionnaire with questions about their expertise in the topics related to the study. They also received training to introduce the main topics. In the second phase, we obtained quantitative data by conducting the original (first and second trials) and new study (third trial). The students of all the experiments were divided into two groups in order to evaluate the performance by executing the review using or not our approach. Finally, in the third phase, the participants of the experiments gave us feedback on the execution of the experiment.

[5] https://doi.org/10.5281/zenodo.3966542.

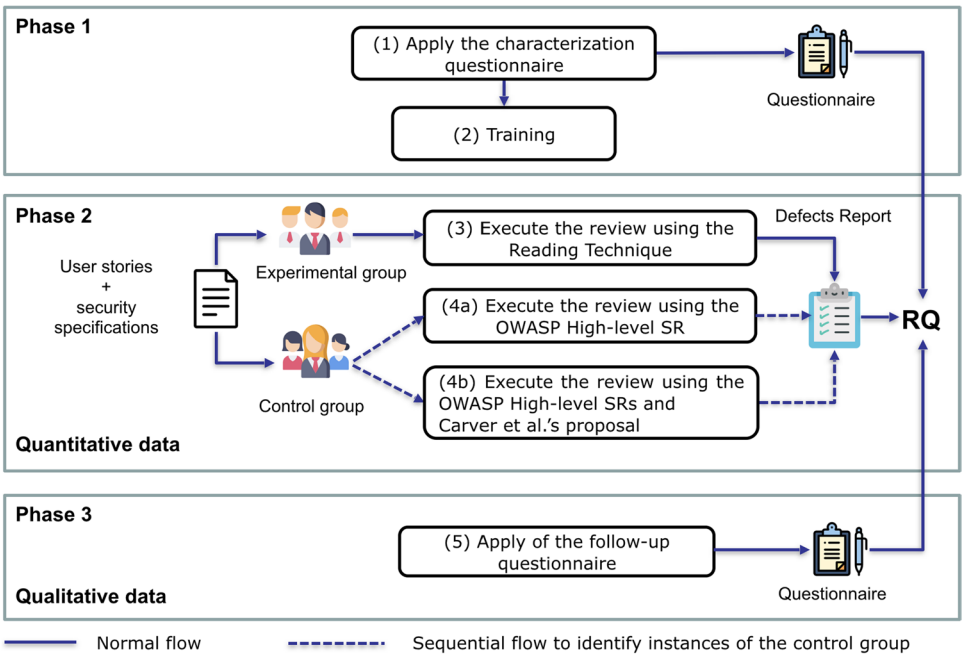

Fig. 7 Experimental design of the study

The instruments used to conduct the experiment are further described in the following.

5.6 Instrumentation

Characterization questionnaire The goal of the questionnaire is to characterize the experience of the subjects and identify key characteristics of four topics: agile software development, RE, software security and inspections.

Training In the first and second trials, the training was focused on the security properties, the OWASP high-level SRs and the defect types involved in the experiment. In the third trial, we included training on how to use each of the inspection techniques. Furthermore, we provided feedback on common errors along with the experiments and we emphasize strictly following the task description of the experiment.

User story and security specifications The input of our approach is a user story and its set of security specifications. Therefore, we created the specifications following the guidelines proposed by Lucassen et al. [35]. They were based on typical customer requests for developing web applications, e.g., sending sensitive information to other systems and deleting data. When doing so, we relied as an orientation on SR specifications from real industrial software projects as used by our industry partners. We did not use real specifications verbatim in our setting for confidentiality reasons.

Task description This document explains to the students how to fill out the defect reporting form according to their treatment. Both treatments received the same specifications. For one treatment (experimental group), the technique was generated according to the user story and its related OWASP high-level SRs. For the other treatment (control group), in the first two trials, the list of the OWASP high-level SRs with the list of the defect types was provided. In the third trial, the control group also received the PBR Black Hat approach.

Defect reporting form This form was used by subjects to record the start and end time of the review, as well as the defects by location and type. The defect reporting form for the experimental group was the one generated for applying the reading technique in Table 8.

Follow-up questionnaire This questionnaire was based on the technology acceptance model (TAM). TAM has been extensively used in several studies [54]. We wanted to know whether the approach was useful and easy to use. We included open text questions to gather feedback about the difficulties and benefits of using our approach.

5.7 Data collection procedure

We used the following procedures to collect the data necessary for answering our research questions and test our hypotheses.

Collect defects found and false positives Inspectors applied the reading technique generated by our approach on the given requirement artifacts, which generated a list of detected defects by each inspector. The same process was applied by the remaining treatments involved in the experiments. We reviewed the output of each inspector in pairs of researchers and classified a registry as false positive if it was

not a defect. Based on these data, we evaluated the performance of the treatments in terms of effectiveness, considering only defects (i.e., true positives).

Collect time spent for detecting defects In this study, time spent refers to the amount of time inspectors spent to identify defects. We collected the time spent by each inspector during the inspection. The time spent is the difference between the start time and end time. Note that this time does not include activities such as training and follow-up questionnaire. We defined 1 hour as the maximum time limit to be spent by an inspector to find the seeded defects. The pilot study was helpful to define the time it would take for inspectors to complete the inspection task. Based on these data, we evaluated the performance of the treatments in terms of efficiency.

Collect perceptions of the subjects Finally, to answer RQ2, we used the TAM based [54] follow-up questionnaire to collect feedback on the usefulness and ease of use of our approach and the PBR Black Hat approach.

5.8 Analysis procedure

We structure our analysis procedure into four steps. Each step leads to the results necessary for answering one of our research questions.

Calculate effectiveness and efficiency of the experimental treatments First, we analyzed the performance of each treatment by carrying out descriptive statistics such as the percent of defects detected in the agile specifications and the time duration to perform the review. Hypotheses testing was also applied. To this end, the analysis was conducted using the statistical tool RStudio version 1.1.4. For the hypotheses testing, we used the Mann–Whitney test with alpha = 0.05. The small number of independent samples motivated this choice of statistical significance and test. Second, to get a deeper insight into the defects detection, we analyze the distribution of types of defects identified by inspectors. This provided answers for RQ1.

Analyze the number of false positives We know that the reliability of an inspection technique is important. Therefore, we analyze false positives to determine to which extent our approach leads inspectors to false positives. We also analyzed this metric for the approaches compared in the experiments.

Interpret questionnaire answers We analyze the frequencies of responses to the TAM questionnaire. Additionally, we conducted a qualitative analysis applying grounded theory open coding activities to the open questions. This provided answers to RQ2.

5.9 Operation of the experiments

The original study (i.e., the first two trials) was conducted in March of 2019 [57], whereas the new study (i.e., the third trial) was conducted in November of 2019. All trials were executed along two days. On the first day, the subjects answered the characterization form in order to allow dividing them into experimental groups. On the second day before the execution of the experiment, concepts of the security properties, OWASP high-level SRs and defect types were reviewed by subjects in a training session. In the case of the new study, training on the inspection techniques was provided. After that, the inspection was conducted as follows.

All subjects had up to one hour to finish the review. In the original study, the control group used the OWASP high-level SRs as support during the review, while in the new study the control group used the PBR Black Hat approach. The experimental group used our approach. When the subjects finished the task, they had to fill out the follow-up questionnaire.

6 Results

In the following, we present the results of the trials conducted in the study. We first describe the results on defect detection effectiveness and efficiency. We also present more specific results on false positives introduced by inspectors and the types of defects identified by them. We end by evaluating the perception of the inspectors on the usefulness and ease of use of our approach.

6.1 Results on defect detection effectiveness

In this section, we present the performance of inspectors in terms of effectiveness across the trials involved in the experiments. We analyze the number of defects found by inspectors who used our approach and other inspection methods with different levels of support. We wanted to understand the potential of our approach to detect security-related defects in agile requirements specifications of web applications. For this, we compared the performance of the effectiveness of our approach against the performance of the effectiveness of inspections conducted with the OWASP high-level SRs and the PBR Black Hat approach. Figure 8 presents the defect detection effectiveness of each inspection technique by showing the distribution of the number of defects found by the subjects in each experimental trial.

It is possible to observe that our approach was more effective than inspections conducted with the OWASP high-level SRs and the PBR Black Hat approach. In the original study (first and second trials), both experimental groups (green block) identified more defects than the control groups (blue block). For instance, in the first trial of the original study,

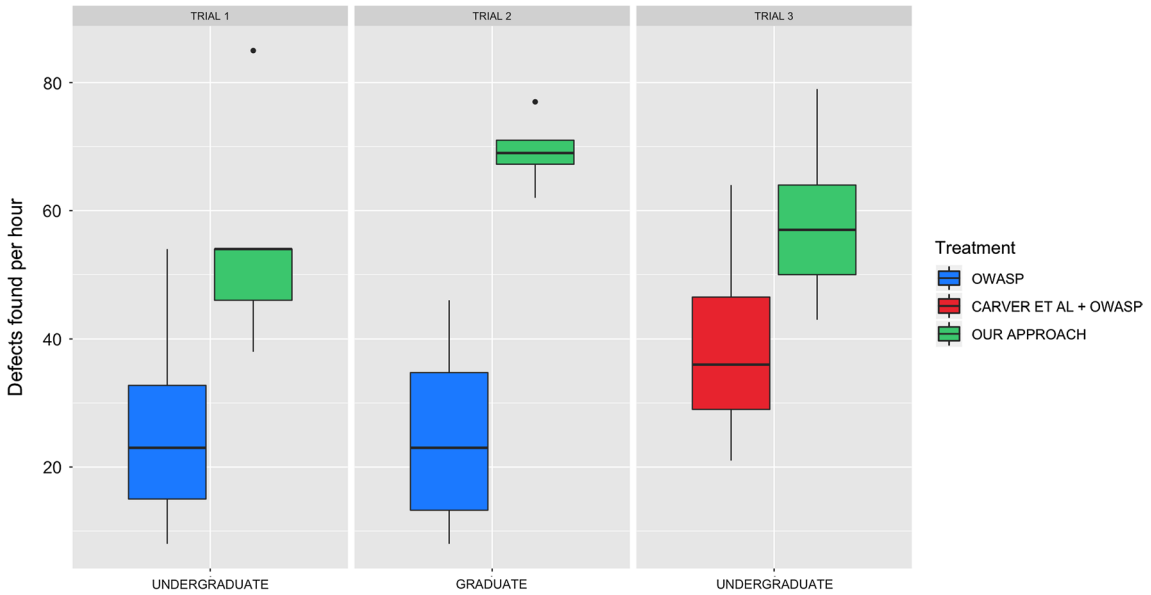

Fig. 8 Defect detection effectiveness across the experiments

conducted by undergraduate students, the experimental group (students who used our approach) identified in median, 54% defects, while the control group (students who conducted the inspection by using the OWASP high-level SRs) identified 23%. The difference was even higher when observing the performance of the second trial conducted by graduate students. Those who used our approach identified, in median, 69% of the defects versus 23% identified by the students who did not use it. This improvement can be explained for two reasons: (1) Often, graduate students have more experience than undergraduate students since the first ones have faced several real projects. We showed this trend in our characterization questionnaire. On the other hand, (2) in the second trial, we slightly modified the defect reporting form to ease its understanding and fulfillment. The reason was that in the first trial several inspectors mentioned that the defect reporting form was confusing. The change consisted of merging the columns A, IS and IR to understand better that those defect types do not have a 1 to 1 relationship with the security specifications such as the O column. In other words, we improved the design of the defect reporting form, while it remained to capture the same information.

Concerning the third trial (new study) conducted by undergraduate students, the effectiveness followed the same pattern as the original study. That is, students who used our approach identified more defects than students who used the PBR Black Hat approach. Note that in this trial, the control group also used the OWASP high-level SRs to support the PBR Black Hat approach. We decided to provide that support because the PBR Black Hat approach does not cover all the defect types introduced in the security specifications. It is noteworthy that the PBR Black Hat approach was not originally designed for the agile context; however, at the same time, its conception does not exclude this type of requirements. In the end, the experimental group (green block) identified, in median, 58% of the defects, while the control group (red block) identified 38%.

When comparing the results between the new and the original study, we found that the PBR Black Hat approach (red block) improved the performance in terms of effectiveness when compared to the inspection conducted using only the OWASP high-level SRs (blue block). This indicates that the PBR Black Hat approach also helps in detecting security-related defects. This is outstanding if we consider that the PBR Black Hat approach was conceived in 2002 under different security concerns than those evaluated in our experiment.

We also wanted to test our null hypothesis on the effectiveness (H_{01a} and H_{01b}), i.e., we checked whether the differences obtained in our experiments were significant to affirm or reject the hypotheses. To this end, we used the Mann–Whitney Test. In that sense, the results of the tests allowed to reject H_{01a} and H_{01b} for all experimental trials because we obtained p values of 0.002, 0.012 and 0.004 for the first, second and third trials, respectively. This means there is a significant difference in terms of effectiveness when using our approach compared to conducting inspections with the OWASP high-level SRs and the PBR Black Hat approach. In other words, the amount of defects found of our approach can be considered significantly high compared

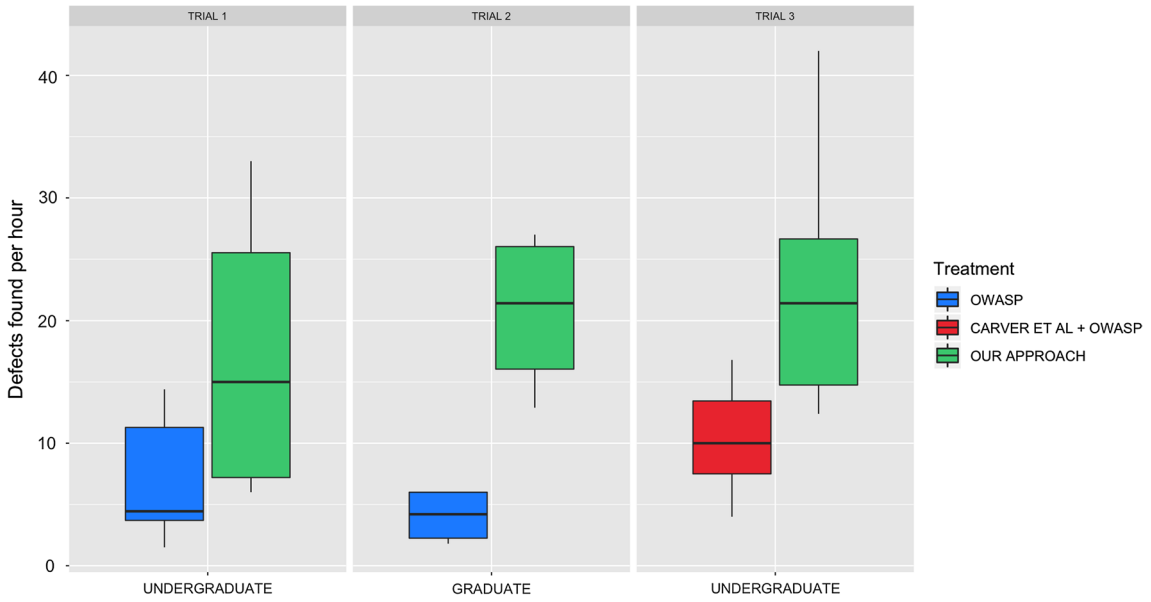

Fig. 9 Defect detection efficiency across the experiments

to the other techniques involved in the experiments. Besides, to complement the p values results, we calculated Cohen's effect size which is a quantitative measure of the magnitude of a phenomenon. For all the experiments, we obtained values of 3.46, 2.24 and 2.47 for the first, second and third trials, respectively. According to [55], this values can be considered as very large effect size. Thus, we can partially answer RQ1: Our approach has a positive impact on defect detection effectiveness with a very large effect size.

6.2 Results on defect detection efficiency

After knowing the effectiveness of our approach across the different trials, we can question its efficiency by analyzing the defects found per hour by the inspectors. Figure 9 shows the distribution of the efficiency of the subjects involved in the experiments.

Note that the efficiency follows the same pattern of the effectiveness; that is, the number of defects found per hour by the students who used our approach (green block) was higher than the students who used the OWASP high-level SRs and the PBR Black Hat approach (blue and red blocks, respectively).

In the original study, both experimental groups, in median, identified more defects per hour than the control groups. For instance, in the first trial conducted by the undergraduate students, our approach efficiency was 15 defects found per hour. (We seeded 14 defects, but participants took less than one hour to complete their tasks.) In contrast, the median of the undergraduate students who used only the OWASP high-level SRs was four. In the second trial conducted by the graduate students, the performance of our approach improved. The mean of our approach efficiency increased to 21 defects found per hour versus four defects found per hour by the graduate students who did not use it. This improvement is proportional to the effectiveness performance shown in Fig. 8. Thus, we can explain these difference across the first two trials.

In the third trial (new study) conducted by undergraduate students, the experimental group identified, in median, 22 defects per hour, whereas the control group identified, in median, ten defects. This means that the inspectors who used our approach identified defects faster than inspectors who used the OWASP high-level SRs and the PBR Black Hat approach. In this trial, we can see that undergraduate students who used our approach performed at the same level than graduate students (second trial) who used our approach. Under the same conditions, this would not be a typical performance considering the experience of the inspectors of each group. We believe that the training provided in the third trial on how our approach works and how the defect reporting form must be filled out may be a cause of this improvement.

Regarding statistical hypothesis testing for efficiency, we found that the Mann–Whitney Test suggests rejecting our second null hypotheses (H_{02a} and H_{02b}) with p values of 0.02, 0.01 and 0.02 for the first, second and third trials, respectively. This means there is a significant difference in terms of efficiency when using our approach against the inspection conducted with the OWASP high-level SRs and the PBR Black Hat approach. In the same direction, we

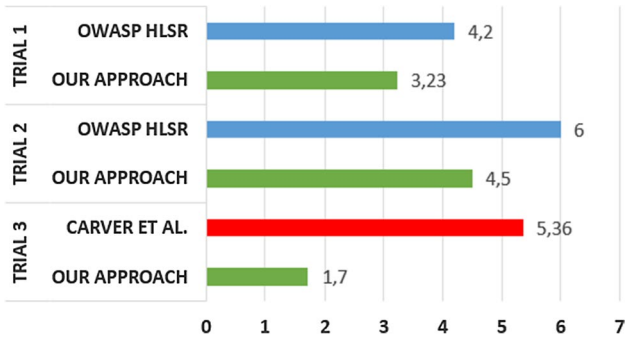

Fig. 10 Average of false positives by trial and technique

found that the relevance of this difference (Cohen's effect size [55]) was large for all the experiments (1.56, 3.29 and 2.32 for the first, second and third trials, respectively). With this information, we can fully answer RQ1. Our approach has a positive impact on security defect detection effectiveness and efficiency when compared to directly using the OWASP high-level SRs and to using the PBR Black Hat approach.

6.3 Results on false positives

To understand to which extent our approach leads inspectors to report defects that are not truly defects (false positives), we analyzed the output of each inspector of the trials to manually and in pairs classify whether the reported defect concerns a true defect or a false positive. False positives may affect both efficiency and effectiveness; thus, these results provide additional insights to answer RQ1. Note that this analysis was conducted in pairs of researchers. We present Fig. 10 to provide an overview of the false positives by trial and groups involved in the experiments.

According to Table 10, we see that all the trials followed the same pattern. Inspectors who used our approach introduced less false positives than inspectors who conducted the inspection by using the OWASP high-level SRs and the PBR Black Hat approach. For instance, in the second trial our approach led the inspectors to introduce, in average, 4.5 false positives while the inspectors who used only the OWASP high-level SRs introduced in average 6.0. In the third trial, where we evaluated our approach against the PBR Black Hat approach, we see that our approach led the inspectors to introduce, in average, 1.7 false positives against 5.4 introduced by the inspectors who used the PBR Black Hat approach. When investigating the possible causes of these findings, we believe that the context factors involved in each experiment and are presented in Table 16, influenced to improve the results of the third trial with respect to the first and second one. For instance, we provided additional training in the new study by teaching the inspectors to follow the task description and our reading technique. Overall, this can be seen as a benefit of our approach since this factor is closely tied to software quality assurance. In other words, if we link these results on false positives with the results on defect detection effectiveness and efficiency, when using our approach inspectors tend to find more defects in less time, making less mistakes. Therefore, these results support again our findings on effectiveness and efficiency presented before.

Table 17 Overall mean scores of performance sorted by efficiency

Ranking	Efficiency (def/h)	Effectiveness	Level	Treatment	Trial
1	22	58.4	U	Our approach	3
2	21	68.9	G	Our approach	2
3	15	54.2	U	Our approach	1
4	10	38.3	U	PBR Black Hat approach	3
5	4	23.6	G	OWASP high-level SRs	2
6	4	23.1	U	OWASP high-level SRs	1

6.4 Analysis across experimental trials

We present a synthesis of our findings from the first, second and third trials conducted in this study as follows. We provide overall median scores for effectiveness and efficiency performance in Table 17. In order to provide a high-level overview of our findings, we rank the results by efficiency performance. We consider efficiency instead of effectiveness given the importance of time spent on agile methods. We also present the subject level, undergraduate (U) or graduate (G), the treatments and the experimental trial number.

If we compare the performance across the trials, we see that the best three performances were obtained by the inspectors who used our approach. In other words, experimental group always performed better than control group. Taking a look at the performance by educational level (undergraduate and graduate), we see that this factor had limited influence on the performance of the inspectors. In our experiments, performance was mainly defined by the treatment given to inspectors; that is, if a support such as structured technique is provided, the performance of the inspectors, regardless of his/her education level, can improve. This is supported by analyzing the performance of the inspectors who used the PBR Black Hat approach. These inspectors performed better than inspectors who did not received support in form of a structured technique.

Table 18 Detail of the inspectors' performance in all trials

ID	Trial	Treatment	Time	OM	%	AM	%	IS	%	IF	%	∑
1	1	Our approach	00:55	4	100	2	50	1	25	0	0	7
2	1	Our approach	01:00	4	100	1	25	1	25	0	0	6
3	1	Our approach	00:48	4	100	0	0	0	0	0	0	4
4	1	OWASP HLSR	00:50	3	75	1	25	0	0	0	0	4
5	1	OWASP HLSR	00:38	3	75	1	25	3	100	0	0	7
6	1	OWASP HLSR	00:44	2	50	1	25	0	0	0	0	3
7	1	Our approach	00:40	3	75	1	25	2	50	0	0	6
8	**1**	**OWASP HLSR**	**00:40**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**50**	**1**
9	1	OWASP HLSR	00:30	0	0	1	25	0	0	1	50	2
10	1	Our approach	00:50	4	100	0	0	0	0	1	50	5
11	1	Our approach	00:20	4	100	0	0	0	0	0	0	4
12	1	OWASP HLSR	00:35	0	0	1	25	1	25	0	0	2
13	1	Our approach	00:36	3	75	0	0	0	0	0	0	3
14	1	OWASP HLSR	00:31	0	0	0	0	2	50	0	0	2
15	**1**	**OWASP HLSR**	**00:28**	**0**	**0**	**0**	**0**	**1**	**25**	**0**	**0**	**1**
16	1	Our approach	00:20	4	100	3	75	2	50	2	100	11
17	1	OWASP HLSR	00:25	1	25	2	50	2	50	0	0	5
18	1	Our approach	00:26	2	50	0	0	0	0	0	0	2
19	1	Our approach	00:17	4	100	1	25	2	50		0	7
20	1	OWASP HLSR	00:25	0	0	1	25	2	50	0	0	3
21	1	OWASP HLSR	00:25	0	0	2	50	3	100	1	50	6
22	1	Our approach	00:20	4	100	0	0	0	0	0	0	4
23	1	Our approach	00:20	4	100	1	25	2	50	0	0	7
24	1	Our approach	00:15	4	100	1	25	2	50	0	0	7
25	1	OWASP HLSR	00:15	2	50	0	0	1	25	0	0	3
26	**2**	**OWASP HLSR**	**00:33**	**0**	**0**	**0**	**0**	**1**	**25**	**0**	**0**	**1**
27	2	Our approach	00:35	4	100	2	50	2	50	2	100	10
28	2	Our approach	00:21	4	100	3	75	2	50	0	0	9
29	2	Our approach	00:20	4	100	3	75	2	50	0	0	9
30	2	OWASP HLSR	00:50	0	0	0	0	2	50	0	0	2
31	2	Our approach	00:37	4	100	1	25	2	50	1	50	8
32	2	OWASP HLSR	01:00	0	0	3	75	2	50	1	50	6
33	2	OWASP HLSR	00:40	1	25	1	25	1	25	1	50	4
34	3	Our approach	00:10	2	50	2	50	2	50	1	50	7
35	3	Our approach	00:15	4	100	2	50	3	75	0	0	9
36	3	PBR Black Hat	00:30	1	25	1	25	2	50	1	50	5
37	3	PBR Black Hat	00:24	0	0	1	25	2	50	1	50	4
38	3	Our approach	00:16	3	75	1	25	3	75	1	50	8
39	3	Our approach	00:18	3	75	1	25	2	50	1	50	7
40	3	PBR Black Hat	00:25	1	25	3	75	3	75	0	0	7
41	3	PBR Black Hat	00:26	0	0	0	0	3	75	1	50	4
42	3	Our approach	00:28	4	100	2	50	3	75	1	50	10
43	3	Our approach	00:26	2	50	2	50	2	50	0	0	6
44	3	PBR Black Hat	00:30	2	50	2	50	2	50	1	50	7
45	3	PBR Black Hat	00:28	4	100	0	0	2	50	0	0	6
46	3	Our approach	00:25	4	100	2	50	2	50	1	50	9
47	**3**	**Our approach**	**00:30**	**1**	**25**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
48	3	Our approach	00:30	2	50	1	25	3	75	1	50	7
49	3	Our approach	00:30	3	75	1	25	2	50	2	100	8
50	3	Our approach	00:31	2	50	1	25	3	75	2	100	8
51	3	PBR Black Hat	00:36	1	25	0	0	2	50	1	50	4

Table 18 (continued)

ID	Trial	Treatment	Time	OM	%	AM	%	IS	%	IF	%	∑
52	3	PBR Black Hat	00:36	4	100	2	50	2	50	1	50	9
53	3	PBR Black Hat	00:43	0	0	2	50	3	75	1	50	6
54	3	PBR Black Hat	00:45	0	0	0	0	3	75	0	0	3
55	3	PBR Black Hat	00:40	1	25	2	50	1	25	0	0	4
56	3	Our approach	00:53	4	100	3	75	3	75	1	50	11

OM Omission, *AM* Ambiguity, *IS* Inconsistency, *IF* incorrect fact

Table 19 Distribution of defects found per treatment

Treatment	Omission (%)	Ambiguity (%)	Inconsistency (%)	Incorrect fact (%)	Total (%)
OWASP high-level SRs	23.2	25.8	36.5	14.8	23
PBR Black Hat + OWASP	32.8	29.5	54.8	31.8	40
Our Approach	92.4	45.2	60.5	42.6	60

6.5 Results on types of defects identified

We also wanted to determine to what extent our approach helps to identify incomplete, inconsistent, incorrect and ambiguous security-related aspects. Table 18 shows the distribution of the defects found per type by each participant in all the trials. We highlighted the discarded subjects as mentioned in Sect. 5.4. For a better understanding of the data shown in Table 18, we summarize in Table 19 the average percentage of defects found by defect type according to the techniques involved in the experiments.

We observed that inspectors who used our approach on average identified 92.4% of omission defects. That is, almost all such defects were identified. Regarding the other defect types, we observed that these inspectors found 45.2% of ambiguity defects, 60.5% of inconsistency defects and 42.6% of defects related to incorrect facts. In total, inspectors who used our approach in average identified 60% of the seeded defects. In comparison with the performance of the inspectors who used the PBR Black Hat, they did not perform at the same level as inspectors who used our approach. For instance, inspectors who used the PBR Black Hat approach found in average 32.2% of omission defects, 29.5% of ambiguity defects, 54.8% of inconsistent defects and 31% of incorrect facts defects. Considering all defect types, in average, this group found 40% of the defects against 60% found by subjects using our approach. The landscape is even poorer if we compare the performance of the inspectors who conducted the inspection by using only the OWASP high-level SRs, which in average found only 23%.

Given these results, we are confident that our approach helps identifying omission defects, that is, to detect security-related aspects that were not considered or were not covered by the security specifications originally created by requirement analysts. We also consider that our approach contributes identifying inconsistency defects since more than half of these defects were identified. Comparing the other defect types, we see that our approach is slightly better than the PBR Black Hat approach and the inspection conducted with only the OWASP high-level SRs.

Note that the results of the ambiguity defect are not very different among the experiments; that is, neither the support of our technique nor the support of the PBR Black Hat approach generates a relevant improvement compared to an ad hoc inspection (using only OWASP high-level SRs).

Despite promising results in identifying defects of omission and inconsistency, we believe that the verification questions that are part of our approach should be reviewed to improve its effectiveness. This is especially important for ambiguity and incorrect fact defects. It is also particularly interesting that the percentage to identify incorrect fact defects was low in all the inspections. Probably, this happened due to the additional domain knowledge that is usually needed to detect such problems. This may indicate the difficulty of identifying this type of defects is higher than the others. In summary, we must consider improvements to increase the effectiveness in detecting this kind of defects.

6.6 Perception of the inspectors on the usefulness and ease of use

After inspectors conducted the inspection task by reviewing the security specifications provided in the experiments, we asked whether they found our approach useful and easy to use. Through the TAM questionnaire [15], we wanted to know about their perceptions on using our approach and the PBR Black Hat approach. Figures 11 and 12 show the frequencies of the responses of the inspectors that measures their perception on ease of use and usefulness, respectively. These figures present green tones that indicate agreement

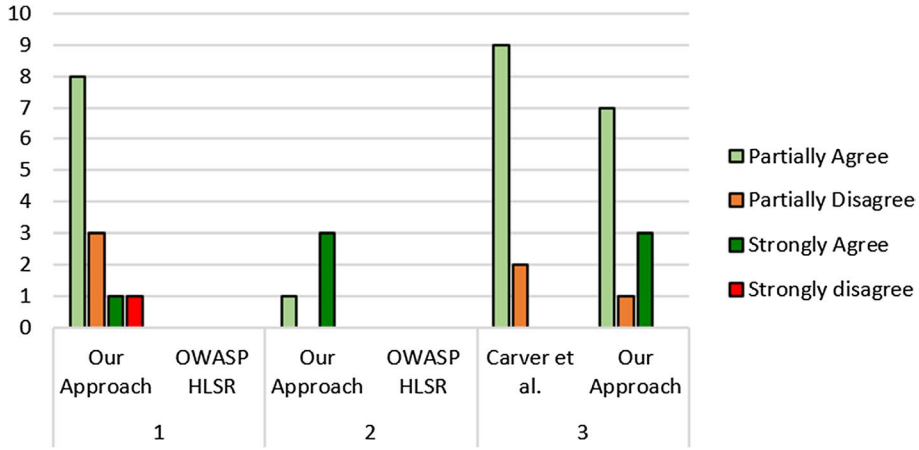

Fig. 11 Perception of the inspectors to the statement "I found the approach easy to use"

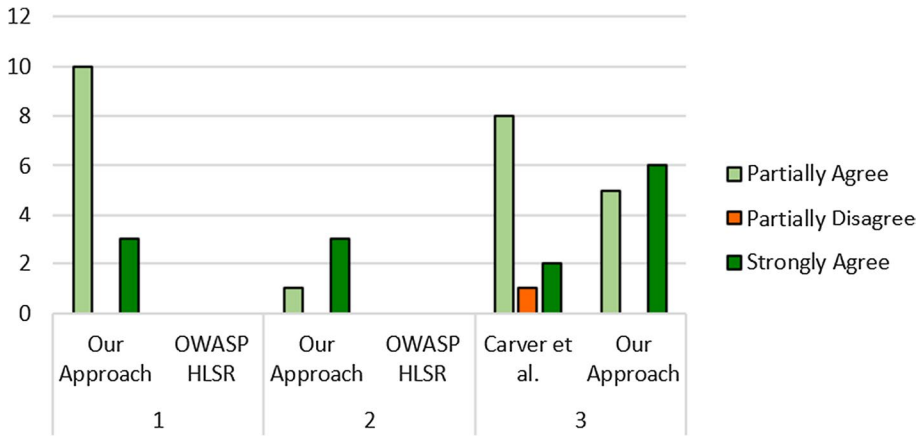

Fig. 12 Perception of the inspectors to the statement "Using the approach improved my performance (find defects faster)"

and red tones that indicate disagreement with our statements of ease of use and usefulness. Note that dark tones indicate a strong agreement/disagreement and light tones indicate a partial agreement/disagreement.

We start by analyzing the responses of the inspectors that measures their perception related to the ease of use of the reading-based approaches. Figure 11 shows the perception of the inspectors for the statement "I found the approach easy to use."

First, we can see that we have a higher concentration of red tones in the first trial. In this case, the perception of inspectors about the ease of use improved across the trials. For instance, in the first trial, 4 out of 13 (31%) strongly or partially disagreed with the statement "I found the approach easy to use." Only one inspector strongly agreed with. In the second and third trials, the picture was different. Just one out of 15 (6%) inspectors who used our approach in these trials partially disagreed. The other 14 inspectors partially (8) and strongly (6) agreed. A large number of disagreements and partial agreements in the first trial indicated that improvements should be added to facilitate the use of the approach and then improve the detection of defects. In that direction, the inspectors proposed some points that could enhance the ease of use of the approach, such as providing a lighter document, modifying the design of the defect reporting form and showing an example of how to fill it out correctly. This feedback was taken into account to perform the second and third trials. In fact, as mentioned in Sect. 6.1, the inspectors' defect detection effectiveness improved, in our opinion, because we introduced some changes to the defect reporting form such as merging the columns related to ambiguous, inconsistency and incorrect facts and simplifying the number of steps in the task description document.

When comparing the perception of ease of use of the approaches involved in the third trial (our approach against the PBR Black Hat approach), we see that, in principle, inspectors found our approach easier to use. For instance, nine of the inspector who used the PBR Black Hat approach partially agreed and none of them strongly agreed with the sentence about ease of use. This indicates they faced difficulties to follow the inspection under its guidance. In contrast, seven inspectors who used our approach partially agreed and three strongly that the approach was easy to use.

We were also interested in knowing the perception of the inspectors in relation to the usefulness of the approach. We defined usefulness as a metric of productivity based on

efficiency. More specifically, we asked whether using the approach would improve their performance when conducting the security requirement inspection (find defects faster). Table 12 gives an insight into the perception of the inspectors related to the usefulness of our approach.

In this case, the perception of the inspectors regarding the usefulness of our approach in the trials was, in general, positive. For instance, all the inspectors who used our approach (28) partially or strongly agreed with the sentence we provided about usefulness. More specifically, 12 out of them (43%) strongly agreed and 16 (57%) partially agreed. However, note that most of the partial agreements were reported in the first trial (10 out of 16). This perception arises from the difficulties faced by the inspectors in that trial. For example, one inspector stated the following: "The review may be exhausting and time-consuming because the task description document is not lightweight." Another one stated: "It would be better to automate the proposal." The inspectors also mentioned some difficulties faced, such as "The verification questions could indicate better how to identify defects such as ambiguity or incorrect facts." We believe these barriers may affect the performance of the inspectors and, thus, the perception of usefulness.

Concerning the perception of the inspectors who used the PBR Black Hat approach in the third trial, we observed that most of the inspectors found it useful. In this case, just one inspector partially disagreed with the sentence about the usefulness we provided. Eight out of 11 (73%) partially agreed and 2 out of 11 (18%) strongly agreed. This reinforces the idea that novice inspectors need support to review security-related aspects. Therefore, approaches addressing these concerns help to identify defects that would not be easily identified by ad hoc inspections.

Nevertheless, if we compare the perception of usefulness between our approach and the PBR Black Hat approach, we see that more inspectors strongly agreed with our approach. This reflects that inspectors felt more comfortable using our approach. Also, this perception is aligned with the performance of inspectors across the trials, since inspectors who used our approach performed better than inspectors who did not use it.

In summary, to the question of how do the inspectors perceive the usefulness and ease of use of our approach, we have that inspectors who used our approach found it, in principle, easy to use and usefulness. Indeed, the perception of the inspectors was positive in comparison with other security inspection techniques. However, we have several challenges to improve these aspects, such as automating our approach and improving the security verification questions.

7 Threats to validity

We report several threats to validity following the recommendations by Wohlin et al. [58] that we considered or mitigated during the design and execution of the original and new study.

7.1 Internal validity

First, aiming to avoid personal bias, we used researcher triangulation to collect and analyze all data. The profile of researchers varies in relation to experience, but remains in relation to the application domain, in this case, software engineering. One master and one Ph.D. student in informatics with the supervision of one senior researcher were involved in this research triangulation. We carefully reviewed the extraction of defects and calculation of the percentages of performance of the inspectors.

Second, we characterized all the subjects with the aim of removing confounding factors. The characterization allowed us to apply the blocking principle by distributing the participants so that these characteristics were equally distributed among the experimental and control groups. Table 12 highlights the subjects who were chosen because of the partition principle. Random assignment was employed for subjects with similar characteristics.

We also consider that the performance of participants could be affected if inspectors try to guess the purpose of the experiment. In the original study, participants were not aware of the existence of experimental versus control groups or whether they belonged to different groups. Participants were told only that they were supposed to perform the task of detecting security-related defects based on a given requirement specifications. Thus, single blinding was used to minimize biases. Regarding the new study, participants were aware of the group treatment because we introduce them both the inspection techniques involved in the experiment. We examined the responses by the groups to see if they resembled closely with treatment responses. However, we did not find any evidence of treatment diffusion across the groups.

Finally, regarding training, we provided the same examples of user stories, defects, security specifications and controls to the experimental and control group of all the trials, so any potential bias is similar for all the subjects. In summary, participants received the same training.

7.2 Construct validity

In our evaluation, we analyzed the suitability of our approach in terms of the number of defects detected, defect types and

number of false positives. To this end, quantitative analysis was performed. We used metrics such as effectiveness and efficiency that are commonly used in inspection studies that are empirically evaluated. We also analyzed the perceptions on usefulness and ease of use of the inspectors when using our approach and the PBR Black Hat approach. To this evaluation, qualitative analysis was considered. We used the TAM questionnaire [15], which has also been widely used and evaluated [54] to measure the acceptances of techniques, applications and technologies.

7.3 Conclusion validity

Reliability of measures is an important consideration to draw valid conclusions about the results. We used the Mann–Whitney Test with the aim of determining whether we reject or not our null hypotheses. The decision of using this method was supported according to the distribution of our independent samples, in this case, the treatments of the experiments. Besides statistical significance, we used the Cohen's *h* metric that is a measure of distance between two proportions or probabilities. With this, we determine whether the difference of our results can be considered as small, medium or large. In our study, we found the relevance scores of the experimental group was significantly better than the control group.

Regarding number of participants, we had 56 participants divided into groups that characterize different samples. Factors such as type of study (original and new), position of the students (undergraduate and graduate) and type of treatment (experimental and control) were considered among the experiments.

7.4 External validity

To mitigate this kind of validity threat, the sample population should be representative of the population we want to evaluate. Regarding the subject representativeness, we used students to represent novice inspectors. Using students as subjects remains a valid simplification of real-life settings needed in laboratory contexts [22]. In the studies, participants are representative of students in computer science enrolled in two different graduate and undergraduate courses. For all of them, we provided concepts related to security principles and inspection techniques. Regarding the objects, we created the agile specifications following the quality guidelines proposed by Lucassen [35]. We also peer-reviewed the requirement specifications and the seeded defects in terms of their representativeness. Through the new study, we have demonstrated the applicability of the requirement specifications created in identifying security-related defects. As we planned to conduct a limited amount of trials with a limited amount of subjects, the experimental package is available for external replications.

8 Discussion

This work brought up several further questions that have strong implications on future research. Therefore, in the following, we discuss several of these aspects in more depth.

8.1 Suitability of the OWASP high-level SRs

We are aware that not all OWASP high-level SRs might be useful in all situations. However, we are confident that as a starting point it is useful to have a basis that allows novice inspectors, at least, to consider the basic needs to deal with security.

8.2 Generalization of our approach

We know that not only security is challenging in agile projects. Indeed, it seems that other NFRs such as maintainability and performance are often ignored or ill-defined in this context. Moreover, plan-driven software projects may face similar problems. This provides an opportunity to extend our approach, e.g., considering other types of inputs such as open textual requirements and covering other quality characteristics such as portability and usability. Currently, knowledge on the available verification techniques to assure these quality characteristics are met is scattered and limited. Furthermore, those techniques are commonly not properly integrated into the agile development philosophy. Thus, we consider this, a first step in this direction was conducted by investigating security, a specific product quality characteristic.

8.3 Implications of our research for practitioners

Unfortunately, security problems tend to be postponed to later stages of software development [47]. For practitioners, our approach provides a way to detect defects related to security aspects that should be specified. According to the results of the original and new study, we are confident to say that in principle our approach supports novice inspectors by providing guidance in applying a reading technique that will help them to identify defects in agile requirements specifications of web applications. In addition, we designed the approach in such a way that it works without expensive review cycles, aligned with the agile philosophy. We see three main potential benefits of this approach: first, narrowing the security knowledge gap that exists between experts and novice inspectors. Second, the reading technique provides a strong focus on security aspects. In this way, the team

can avoid discussing obvious issues and focus on important, difficult, security-specific aspects of the review. Third, we saw that our approach provided positive results regarding the performance of individual inspectors when conducting the security inspection.

8.4 Implications of our research for researchers

This work contributes already to closing an important literature gap that exists with respect to security requirement verification in the agile context. Based on the scarce literature on the topic, we believe that our approach constitutes an interesting starting point to discuss in-depth verification activities centered on security in agile contexts. For us, the results strengthen our confidence in further extending our approach to scale its usability up to practical settings covering a full, tool-supported process integration, which was not (and could not be) in scope of a development in our research-centric environment.

8.5 Limitations

We concentrated on a set of specific security properties and high-level SRs from the OWASP (matching security subcharacteristics also described in the SQuaRE quality model). There are several security standards that are different from the ones provided by OWASP. Thus, we could complement the security vision of our approach with other standards.

Moreover, given the complexity of working with NLP in RE, there is a limitation related to the completeness of the keyword repository needed to link the user stories with the security properties. To deal with this, we decided to consider synonyms regarding the initial set of keywords.

We only evaluated our approach with a use case scenario (two user stories with their security specifications) as a starting point for detecting defects related to security. However, additional use cases can also be provided as input to the participants and may give us insights on the suitability of our approach in different contexts. We are currently designing an industrial case study to evaluate the coverage of detecting security defects by our approach for a real software system. The results will provide evidence on how the approach generalizes when applied with the help of security analysts without the time and other experimental constraints.

We are also aware that the security specifications involved in the experiments constitute a limitation of the study. In a perfect scenario, we would have security concerns specified by companies or independent practitioners, but often this information is restricted. Therefore, we invested our best efforts to carefully create and verify the specifications on their representativeness. Nevertheless, external replications, including a wider range of user stories and security specifications, are needed to improve external validity of our results.

9 Concluding remarks

This work addresses a gap in the literature concerning the absence of verification techniques for security in agile requirements engineering and its lack of empirical evidence. It is well known that the poor definition of NFRs, minimum documentation and lack of requirement verification are among the most important concerns of software requirement engineering researchers [13, 41]. Therefore, we presented an approach for reviewing security-related aspects in agile requirements specifications of web applications, which we empirically evaluated via three controlled experimental trials. In the following, we summarize our conclusions and we discuss potential practical implications of our research.

The three trials concerned evaluating the effectiveness, efficiency, usefulness and ease of use of our approach when compared to an ad hoc inspection supported with the OWASP high-level SRs and another one supported with the PBR Black Hat approach. In the combined analysis of all the controlled experiments, participants in the experimental group performed significantly better than participants in the control group in terms of effectiveness and efficiency, i.e., participants who used our approach identified more defects in less time than participants who did not use it. We also identified that participants who used our approach found it, in principle, useful and easy to use.

Future work includes evaluating the performance of using our approach in industrial settings. We want to reach out to better understand the performance of using our approach in real settings, as well as further information to help us better addressing practitioners needs. Therefore, we might have to provide tool support for the application of our approach, in such a way that, for instance, applying the reading technique could be guided by the FESRAS framework.

References

1. Alsaqaf W, Daneva M, Wieringa R (2017) Quality requirements in large-scale distributed agile projects–a systematic literature review. In: International working conference on requirements engineering: foundation for software quality, pp 219–234. Springer, Berlin
2. Araujo R, Curphey M (2005) Software security code review: code inspection finds problems. Software Magazine. July 2005
3. Azuma M (2001) Square: the next generation of the ISO/IEC 9126 and 14598 international standards series on software product quality. In: ESCOM (European software control and metrics conference), pp 337–346. Springer, Berlin

4. Basili V, Caldiera G, Lanubile F, Shull F (1996) Studies on reading techniques. In: Proceedings of the twenty-first annual software engineering workshop, vol 96, p 002. Citeseer
5. Basili VR (1992) Software modeling and measurement: the goal/question/metric paradigm. Tech. rep
6. Beck K, Beedle M, Van Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R et al (2001) Manifesto for agile software development. http://agilemanifesto.org. Accessed 21 Aug 2020
7. Bjarnason E, Runeson P, Borg M, Unterkalmsteiner M, Engström E, Regnell B, Sabaliauskaite G, Loconsole A, Gorschek T, Feldt R (2014) Challenges and practices in aligning requirements with verification and validation: a case study of six companies. Empir Softw Eng 19(6):1809–1855
8. Boehm B (2002) Get ready for agile methods, with care. Computer 1:64–69
9. Boehm B, Basili VR (2005) Software defect reduction top 10 list. Foundations of empirical software engineering: the legacy of Victor R. Basili 426(37):426–431
10. Cao L, Ramesh B (2008) Agile requirements engineering practices: an empirical study. IEEE Softw 25(1):60–67
11. Carver JC (2010) Towards reporting guidelines for experimental replications: A proposal. In: 1st international workshop on replication in empirical software engineering, pp 2–5. Citeseer
12. Carver JC, Shull F, Rus I (2006) Finding and fixing problems early: a perspective-based approach to requirements and design inspections. STSC CrossTalk
13. Chung L, Nixon BA, Yu E, Mylopoulos J (2012) Non-functional requirements in software engineering, vol 5. Springer, Berlin
14. Daneva M, Wang C (2018) Security requirements engineering in the agile era: how does it work in practice? In: 2018 IEEE 1st international workshop on quality requirements in agile projects (QuaRAP), pp 10–13. IEEE
15. Davis FD (1989) Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS quarterly, pp 319–340
16. Deepa G, Thilagam PS (2016) Securing web applications from injection and logic vulnerabilities: approaches and challenges. Inf Softw Technol 74:160–180
17. Devanbu PT, Stubblebine S (2000) Software engineering for security: a roadmap. In: Proceedings of the conference on the future of software engineering, pp 227–239. ACM, Cambridge
18. Domah D, Mitropoulos FJ (2015) The nerv methodology: a lightweight process for addressing non-functional requirements in agile software development. In: SoutheastCon 2015, pp 1–7. IEEE
19. Eberlein A, Leite J (2002) Agile requirements definition: a view from requirements engineering. In: Proceedings of the international workshop on time-constrained requirements engineering (TCRE'02), pp 4–8
20. Elberzhager F, Klaus A, Jawurek M (2009) Software inspections using guided checklists to ensure security goals. In: 2009 international conference on availability, reliability and security, pp 853–858. IEEE
21. Fabian B, Gürses S, Heisel M, Santen T, Schmidt H (2010) A comparison of security requirements engineering methods. Requir Eng 15(1):7–40
22. Falessi D, Juristo N, Wohlin C, Turhan B, Münch J, Jedlitschka A, Oivo M (2018) Empirical software engineering experts on the use of students and professionals in experiments. Empir Softw Eng 23(1):452–489
23. Fernández DM, Wagner S, Kalinowski M, Felderer M, Mafra P, Vetrò A, Conte T, Christiansson MT, Greer D, Lassenius C et al (2017) Naming the pain in requirements engineering. Empir Softw Eng 22(5):2298–2338
24. Fernández DM, Wagner S, Kalinowski M, Schekelmann A, Tuzcu A, Conte T, Spinola R, Prikladnicki R (2015) Naming the pain in requirements engineering: comparing practices in brazil and germany. IEEE Softw 32(5):16–23
25. FoxBusiness.com: Biggest cyber attacks in history. Yahoo Finance. https://finance.yahoo.com/news/worst-cyber-attacks-past-10-202226243.html . Accessed 21 Aug 2020
26. Goertzel KM, Winograd T, McKinley HL, Oh LJ, Colon M, McGibbon T, Fedchak E, Vienneau R (2007) Software security assurance: a state-of-art report (sar). Tech. rep., Information assurance technology analysis center (IATAC)
27. Haley C, Laney R, Moffett J, Nuseibeh B (2008) Security requirements engineering: a framework for representation and analysis. IEEE Trans Softw Eng 34(1):133–153
28. Halling M, Biffl S, Grechenig T, Kohle M (2001) Using reading techniques to focus inspection performance. In: Proceedings 27th EUROMICRO conference. 2001: a net odyssey, pp 248–257. IEEE
29. Houmb SH, Islam S, Knauss E, Jürjens J, Schneider K (2010) Eliciting security requirements and tracing them to design: an integration of common criteria, heuristics, and umlsec. Requir Eng 15(1):63–93
30. Howard M, Lipner S (2006) The security development lifecycle, vol 8. Microsoft Press, Redmond
31. Inayat I, Salim SS, Marczak S, Daneva M, Shamshirband S (2015) A systematic literature review on agile requirements engineering practices and challenges. Comput Hum Behav 51:915–929
32. Kraut RE, Streeter LA (1995) Coordination in software development. Commun ACM 38(3):69–82
33. Kuhrmann M, Diebold P, Münch J, Tell P, Garousi V, Felderer M, Trektere K, McCaffery F, Linssen O, Hanser E et al (2017) Hybrid software and system development in practice: waterfall, scrum, and beyond. In: Proceedings of the 2017 international conference on software and system process, pp 30–39. ACM
34. Lami G, Gnesi S, Fabbrini F, Fusani M, Trentanni G (2004) An automatic tool for the analysis of natural language requirements. Informe técnico, CNR Information Science and Technology Institute, Pisa, Italia, Setiembre
35. Lucassen G, Dalpiaz F, van der Werf JME, Brinkkemper S (2015) Forging high-quality user stories: towards a discipline for agile requirements. In: 2015 IEEE 23rd international requirements engineering conference (RE), pp 126–135. IEEE
36. McGraw G (2006) Software security: building security, vol 1. Addison-Wesley Professional, Cambridge
37. Mead NR, Stehney T (2005) Security quality requirements engineering (SQUARE) methodology, vol 30. ACM, Cambridge
38. Mellado D, Blanco C, Sánchez LE, Fernández-Medina E (2010) A systematic review of security requirements engineering. Comput Stand Interfaces 32(4):153–165
39. Mellado D, Fernández-Medina E, Piattini M (2007) A common criteria based security requirements engineering process for the development of secure information systems. Computer standards & interfaces 29(2):244–253
40. Nerur S, Mahapatra R, Mangalaraj G (2005) Challenges of migrating to agile methodologies. Commun ACM 48(5):72–78
41. Nuseibeh B, Easterbrook S (2000) Requirements engineering: a roadmap. In: Proceedings of the conference on the future of software engineering, pp 35–46. ACM
42. OWASP: The Open Web Application Security Project. https://owasp.org. Accessed 21 Aug 2020
43. Peine H, Jawurek M, Mandel S (2008) Security goal indicator trees: A model of software features that supports efficient security inspection. In: 2008 11th IEEE high assurance systems engineering symposium, pp 9–18. IEEE

44. Penzenstadler B, Raturi A, Richardson D, Tomlinson B (2014) Safety, security, now sustainability: the nonfunctional requirement for the 21st century. IEEE Softw 31(3):40–47
45. Ramesh B, Cao L, Baskerville R (2010) Agile requirements engineering practices and challenges: an empirical study. Inform Syst J 20(5):449–480
46. Riaz M, King J, Slankas J, Williams L (2014) Hidden in plain sight: automatically identifying security requirements from natural language artifacts. In: 2014 IEEE 22nd international requirements engineering conference (RE), pp 183–192. IEEE
47. Sampaio L, Garcia A (2016) Exploring context-sensitive data flow analysis for early vulnerability detection. J Syst Softw 113:337–361. https://doi.org/10.1016/j.jss.2015.12.021
48. Schön EM, Thomaschewski J, Escalona MJ (2017) Agile requirements engineering: a systematic literature review. Comput Stand Interfaces 49:79–91
49. Shull FJ, Basili VR (1998) Developing techniques for using software documents: a series of empirical studies. Ph.D. thesis, research directed by Dept. of Computer Science. University of Maryland
50. Slankas J, Williams L (2013) Automated extraction of non-functional requirements in available documentation. In: 2013 1st International workshop on natural language analysis in software engineering (NaturaLiSE), pp 9–16. IEEE
51. Subashini S, Kavitha V (2011) A survey on security issues in service delivery models of cloud computing. J Netw Comput Appl 34(1):1–11
52. Terpstra E, Daneva M, Wang C (2017) Agile practitioners' understanding of security requirements: insights from a grounded theory analysis. In: 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW), pp. 439–442. IEEE
53. Travassos G, Shull F, Fredericks M, Basili VR (1999) Detecting defects in object-oriented designs: using reading techniques to increase software quality. In: ACM Sigplan notices, vol 34, pp 47–56. ACM
54. Turner M, Kitchenham B, Brereton P, Charters S, Budgen D (2010) Does the technology acceptance model predict actual use? a systematic literature review. Inf Softw Technol 52(5):463–479
55. VanVoorhis CW, Morgan BL (2007) Understanding power and rules of thumb for determining sample sizes. Tutor Quant Methods Psychol 3(2):43–50
56. Villamizar H, Kalinowski M, Viana M, Fernández DM (2018) A systematic mapping study on security in agile requirements engineering. In: 2018 44th Euromicro conference on software engineering and advanced applications (SEAA), pp 454–461. IEEE
57. Villamizar H, Neto AA, Kalinowski M, Garcia A, Méndez D (2019) An approach for reviewing security-related aspects in agile requirements specifications of web applications. In: 2019 IEEE 27th international requirements engineering conference (RE), pp 86–97. IEEE
58. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer, Berlin
59. Zubrow D (2004) Software quality requirements and evaluation, the iso 25000 series. Software Engineering Institute, Carnegie Mellon