An Approach for Reviewing Security-Related Aspects in Agile Requirements Specifications of Web Applications

Hugo Villamizar, Amadeu Anderlin Neto, Marcos Kalinowski, Alessandro Garcia
Software Engineering Laboratory
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil
{*hvillamizar, aanderlin, kalinowski, afgarcia*}*@inf.puc-rio.br*

Daniel Méndez
Software Engineering Research Lab Sweden
Blekinge Institute of Technology
Karlskrona, Sweden
daniel.mendez@bth.se

***Abstract*—Defects in requirements specifications can have severe consequences during the software development lifecycle. Some of them result in overall project failure due to incorrect or missing quality characteristics such as security. There are several concerns that make security difficult to deal with; for instance, (1) when stakeholders discuss general requirements in meetings, they are often unaware that they should also discuss security-related topics, and (2) they typically do not have enough expertise in security. 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 involved. The goal of this paper is to design and evaluate an approach for reviewing security-related aspects in agile requirements specifications of web applications. The approach considers user stories and security specifications as input and relates those user stories to security properties via Natural Language Processing. Based on the related security properties, our approach then identifies high-level security requirements from the Open Web Application Security Project to be verified and generates a reading technique to support reviewers in detecting defects. We evaluate our approach via two controlled experiment trials. We compare the effectiveness and efficiency of novice inspectors verifying security aspects in agile requirements using our approach against using the complete list of high-level security requirements. The (statistically significant) results indicate 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; requirements verification; software inspection; software security;**

I. Introduction

Requirements Engineering (RE) is an inherently complex part of software engineering. Misunderstandings and defects in requirements-related artifacts can easily lead to design flaws and cause severe and costly problems [18]. Agile RE relies on lightweight documentation and face-to-face collaborations between customers and developers [6]. Yet, agility does not necessarily compensate the problems of more plan-driven software process models. In fact, it can even make those problems more explicit if a key prerequisite for successful RE is not given: human-intensive exchange, collaboration, and trust [14]. In other words, agile RE has already helped to address some specific problems of RE, but it has also brought others to the surface [24].

Security is an essential Non-Functional Requirement (NFR) that requires special attention, inter alia, due to business needs to protect data. Much of sensitive information is hosted on the internet, making web applications a target. Security requirements often appear throughout but also beyond the elicitation phase, so stakeholders are often simply unaware of them. For instance, when stakeholders make decisions during the meetings, they are often unaware that these decisions might also raise data protection-related issues [18]. This often leads to unspecified security aspects.

However, the picture is even more challenging in an agile context. Several studies have identified problems that could result from the poorly detailed requirements specifications [6], [27]. These problems can result in overall project failure due to incorrect or missing functionalities and/or quality characteristics. According to Eberlein [11], 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 [22]. In that sense, agile RE should include more detailed requirements verification into the process [11], [27].

Despite the apparent disadvantages of integrating security into agile contexts, a recent study by Villamizar et al. [36] identified a lack of research on security requirements verification. Such activities should be conducted to assure that agile requirements specifications are correct, consistent, unambiguous, and complete. This means properly covering security-related aspects, such as task-flow constraints or assignment of administrative privileges.

Given this scenario, our work aims at contributing to closing the literature and industry gaps that exist concerning to security requirements verification in agile contexts. More specifically, our goal is to propose and evaluate an approach for reviewing security-related aspects in agile requirements specifications of web applications. We decided to focus on web applications, given that they have become a common

2332-6441/19/$31.00 ©2019 IEEE
DOI 10.1109/RE.2019.00020

target for accessing information and manipulating or extracting sensible data. To achieve this goal, we defined the following specific goals:

1) Develop an approach for reviewing security aspects in agile requirements specifications of web applications.
2) Evaluate the approach to analyze whether it helps inspectors to review security aspects. To this end, we report on a controlled experiment to observe the impact on effectiveness and efficiency. We also observe the perceived usefulness and ease of use.

As a result of our work, we found that using our approach had a positive impact when reviewing security in agile specifications of web applications. Our results indicate significant differences when comparing the performance of inspectors using our approach versus other defect-based technique.

The remainder of this work is organized as follows. Section II introduces the background on agile RE and how security verification is typically performed in this context. Section III introduces the approach we designed to deal with security verification in agile and early manner. Section IV presents the study design used to evaluate the approach. The results are presented and discussed in Sections V and VI, respectively. Section VII provides a discussion of limitations of our approach before we conclude with Section VIII.

II. Background and Related Work

This section introduces the background on agile and security requirements and describes work related to security requirements verification in agile context.

A. 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 [19]. Yet, not much is known about the challenges posed by the collaboration oriented agile way of dealing with RE activities. Ramesh et al. [28] 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 requirements verifications. The recent report from the NaPiRE initiative [14] extends the challenges with (i) communication flaws between teams and customers, and (ii) 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.

B. Security Requirements (SRs)

Software development should be conducted with security in mind at all stages and it should not be an afterthought [22]. Developing secure software is not a trivial task due to the lack of security expertise in developers and the inadequacy of methodologies to support developers who are not security experts [18]. Yet, in the majority of projects, security is often dealt with in retrospective, when the system has already been designed and put into operation [23].

SRs have traditionally been considered quality requirements [8], [9]. Like other quality requirements, they tend not to have simple yes/no satisfaction criteria. Haley et al. [17] 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 very difficult, if not impossible, to measure negative properties. Second, for SRs, the tolerance on "satisfied enough" is small, often zero, given the implications of noncompliance. Moreover, stakeholders tend to want SRs 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. This is even more challenging in agile contexts because, apparently, functionality is the primary focus in agile, while NFRs are typically either ignored or ill-defined [27].

C. SRs Verification in Agile Contexts

In general, several authors have worked on quality assurance methods for verifying the quality properties of different artifacts. One of the most compared and evaluated methods, in several experiments and studies, are inspection techniques. In that direction, we can name reading support techniques for defect detection such as Perspective-Based Reading (PBR) [3], Defect-Based Reading (DBR) [15] and Use-Based Reading (UBR) [38], which are well known and established. As far as security inspections are concerned, little work has been published (e.g., [7], [10], [12]) on how to support inspectors with detailed reading support for reviewing security-related aspects.

Elberzhager et al. [12] 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 requirements specification and within agile requirement artifacts. Second, our approach addresses high-level SRs as defined by the Open Web Application Security Project (OWASP) [26], 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.

Carver et al. [7] further describe a set of perspectives that provide security-specific questions for a requirements 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 Curpheys article on security code reviews [1]. However, due to the large number of software vulnerabilities and the variety of ways to deploy computer attacks, it is not enough to consider only three types of security controls. Indeed, the list is incomplete when compared to OWASP high-level SRs.

In the agile context, the picture is even poorer. We are aware of only one study that it is part of the results of a recent systematic mapping [36]. Domah et al. [10] propose a lightweight methodology to address NFRs early in agile software development processes. NFRs 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 requirements specifications. Hence, previous knowledge on security is required to take advantage of the methodology. In summary, few approaches exist to address the systematic analysis and detection of security problems, especially during early stages, and the scenario is even worse in agile contexts.

III. Our Approach

In this section, we propose our approach for reviewing security-related aspects in agile requirements specifications of web applications. The approach aims at addressing security in early stages of the software development lifecycle.

A. 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 [20]. In this context, user story is the most frequently used artifact for requirements specification [29]. 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) and current research (e.g., [32]) on web application security use OWASP as a definitive reference. Hence, we consider the reliability of this project as a reasonable assumption.

B. 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 security analysts. It provides them with a reading technique to assist in the identification of defects related to security aspects in agile requirements specifications. According to Nerur [25], people with a high-level of competence are of vital importance in agile teams. Much of the knowledge in agile development is tacit and resides in the heads of the development team members [5]. Nevertheless, in agile contexts, there is a focus on functionality. Competence in software security is typically not widely spread among agile practitioners [16]. Therefore, our approach focuses on providing a detailed reading technique to support novice inspectors. We believe that more experienced security analysts could still use the approach, but they are outside in 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 [26]. These high-level SRs describe the most important security features that architects and developers should include in every web application [26]. The System and Software Quality Requirements and Evaluation (SQuaRE) model [39] 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: (i) confidentiality and (ii) identification and authentication. Table I presents the security properties considered by our approach.

Table I
Security properties considered by our approach

Security Property	Description
Confidentiality (C)	Degree to which the data is 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.

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 [33]. A common defect taxonomy used when inspecting require-

ments is the one proposed by Shull [30]. 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 II shows the defect types covered by our approach and their definitions.

Table II
DEFECT TYPES DEFINITION IN SCOPE OF OUR APPROACH

Defect Type	Definition
Omission (O)	Necessary information about the system has been omitted from the software artifact.
Ambiguity (A)	A requirement has multiple interpretations due to multiple terms for the same characteristic.
Inconsistency (IS)	Two or more requirements are in conflict.
Incorrect Fact (IF)	A requirement asserts a fact that cannot be true under the conditions specified for the system.

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 [2]. 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 lifecycle 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 [4]. Hence, we envision that our approach is used just before a user story is defined as ready for codifying.

C. Overview of our Approach

We propose our approach in two defined phases: (1) generating the reading technique based on the agile requirements specification, and (2) following the reading technique to identify defects. These phases are shown in detail in Figure 1 and explained as follows.

Phase 1: Generate Reading Technique. To generate the focused reading technique, we use Natural Language Processing (NLP) to extract keywords from the user story. Thereafter, these words are used 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

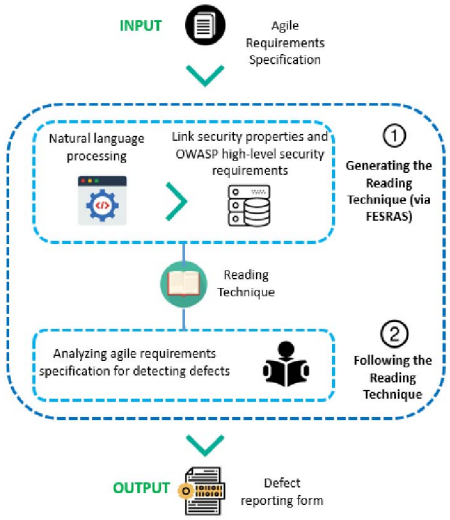

Figure 1. Overall structure of our approach

a key factor for achieving software quality [21]. 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. 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. Therefore, nouns are also extracted for matching purposes. In summary, the verb is extracted from the second block of the user story format and the nouns are extracted from the third block (*cf.* Table III).

Table III
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 extract the words, we developed a Software Framework (FESRAS),[1] which uses the Stanford CoreNLP tool[2] 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. Thereby, the software framework can take the verbs and nouns of the user story and then analyze them to link security properties.

[1]https://github.com/hrguarinv/FESRAS
[2]https://github.com/stanfordnlp/CoreNLP

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. As an example, Table IV shows some keywords that are part of the repository used to indicate which security properties should be considered. Our online material, available at Zenodo, contains all the words of the repository.

Table IV
RELATIONSHIP BETWEEN THE KEYWORDS AND SECURITY PROPERTIES

	Security Property			
Keyword	**Confidentiality**	**Integrity**	**Availability**	**IA**
Access	X			X
Change		X		
Export	X			
Recover			X	
Password				X

This repository is based on a similar one provided by Slankas [31] in their work about automated extraction of NFRs in available documentation. The changes are that we focused on security and therefore we complemented the set of keywords with synonyms and more specific words stated by OWASP. If there is no match between the keywords and the security properties, our approach will link the user story to all the security properties stored in the repository. Table V shows the OWASP high-level SRs by security property.

Table V
OWASP HIGH-LEVEL SRS BY SECURITY PROPERTY

Property	**OWASP High-Level SRs**
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 is 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 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.

For each user story, the reading technique focuses the reviewer to verify whether its security specifications contain any of the defect types. This happens when reviewers check the security specifications against the OWASP high-level SRs of the linked properties. To reach this, the reading technique contains a set of verification questions to help identify the different defect types. Table VI shows the questions.

Table VI
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?

Phase 2: Following the Reading Technique. This phase aims to guide the reviewer in finding such requirements defects. Using the generated reading technique, reviewers can follow the instructions and answer the questions in order to look for defects.

To facilitate the review, our approach rewrites the OWASP high-level SRs in a way so 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., *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.

Furthermore, the security statements also present examples for some concepts in order to give inspectors an idea about the context of the OWASP high-level SRs. An example follows: *I2. All input, e.g., query parameters, string variables and cookies, is validated to be correct and fit for the intended purpose*. That way, reviewers are provided with a reading technique that should increase their performance during the review. Next, we present a motivational example.

D. Motivational Example

In the following, we demonstrate the application of our approach in an example setting. Table VII shows a user story and its set of security specifications with some defects commonly applied to any agile software project.

With the user story in sight, the framework extracts the keywords of the second and third block of the user story, and then matches these keywords related to security properties. In this case, the extracted words are “export”

Table VII
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.	1. The system shall ensure that there is no residual data exposed.
	2. The system shall store credentials securely using the AES encryption algorithm.
	3. The system shall use the RSA encryption algorithm to protect all data all the time.
	4. The system shall deactivate a session when it exceeds certain periods of inactivity.
	5. The system shall encrypt the roles and privileges of the system.

(second block) and "system" (third block). Thereafter, the framework can verify whether some security property is related to the extracted words. According to Table IV, "export" matches confidentiality, while "system" does not match any of the security properties. Therefore, our approach can propose OWASP high-level SRs for the confidentiality security property to be verified for the user story (*cf.* Table V, confidentiality).

Our approach then generates the defect reporting form by showing the user story with its security properties and its OWASP high-level SRs, and the verification questions. Thus, inspectors know which security aspects they should verify. 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 SS. 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 OWASP high-level SRs linked in Table V are related or make sense to the security specifications. This means that two OWASP high-level SRs (C2 and C4) are not covered by the security specifications. Therefore, we have detected two defects that should be marked as "omission".

Does any SS allow for multiple interpretations? 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 SS in conflict? SS2 and 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 SS stating a characteristic that cannot be true under the conditions specified for the system? 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 VIII 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.

Table VIII
DEFECTS REPORTING FORM

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

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 OWASP high-level SRs (C2, C4) are not related to the security specifications. The rest of the defects (4) are related to ambiguous, inconsistent and incorrect fact defects. In this case, SS4 was marked as ambiguous, SS2 and SS3 were marked as inconsistent and SS5 was marked as incorrect.

IV. EXPERIMENT

For a better understanding of the feasibility of using our approach for reviewing security aspects in agile requirements specifications of web applications, we conducted a controlled experiment in academic settings. The choice of a controlled (difference-making) experiment in a laboratory setting is intended rather than being opportunistic, because we are specifically interested in investigating specific phenomena in isolation as preparation for scaling our implementation (and evaluation) up to practice. To this end, we followed the guidelines proposed by Wohlin et al. [37].

A. Goal and Research Questions

We detail the goal of this study in Table IX.

We formulated two Research Questions (RQs). *(RQ1) Does the approach have an effect on defect detection effectiveness and efficiency?* and *(RQ2) How do the inspectors perceive the usefulness and ease of use of the approach?*

B. Experiment Context

The experiment was conducted in two trials, involving students enrolled in Software Engineering classes at the Pontifical Catholic University of Rio de Janeiro. It is noteworthy

Table IX
GOAL-QUESTION-METRIC OF THE EXPERIMENT

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	researcher (on the measured effectiveness and efficiency) and inspectors (on the perceived usefulness and ease of use)
in the context of	novice inspectors using our technique, when compared to using the OWASP high-level SRs and the defect types.

that we also carried out a pilot study with two independent volunteers. The aim was to evaluate the overall (particularly technical) feasibility, time, adverse events, and improve the experiment materials before the experiment trials.

We created the specifications based on typical customer requests for developing web applications, e.g., sending sensitive information to other system and deleting data. When doing so, we relied as an orientation on SRs specifications from real industrial software projects as used by our industry partners. Our goal is to increase the similarity to the studied population units, but did not use real specifications verbatim in our setting for confidentiality reasons. Our SRs specifications contain a set of user stories in this format: As a [*Role*], I want [*Feature*], so that [*Reason*]. The document also contained the related security specifications with seeded defects.

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. Table X shows the distribution of the seeded defects per user story. In total, 13 defects were seeded. The representativeness of the requirements specifications and the defects was reviewed by three independent researchers before conducting the experiment trials.

Table X
DISTRIBUTION OF THE SEEDED DEFECTS

User Story	Omission	Ambiguity	Inconsistency	Incorrect Fact	Total
US1	2	2	2	1	7
US2	2	2	1	1	6

C. Variables Selection

The independent variable in the experiment is the treatment applied by the groups in order to find defects in the SRs specifications. While the control group received OWASP high-level SRs and a list of defect types to be found, the experimental group received our reading technique.

Regarding 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 Section IV-D. We also collected qualitative data with open questions in a follow-up questionnaire. The aim was to gain insights about the perceived usefulness and ease of use of the approach.

D. Hypotheses

Using the variables described in the previous subsections, we defined the following hypotheses.

- **H0a:** There is no difference in terms of effectiveness when using both techniques.
- **H1a:** There is a difference in terms of effectiveness when using both techniques.
- **H0b:** There is no difference in terms of efficiency when using both techniques.
- **H1b:** There is a difference in terms of efficiency when using both techniques.

E. Selection of Subjects

Our subjects were intended to represent novice inspectors. We selected subjects, by convenience, from classes on Software Engineering, involving 25 undergraduate (first trial) and 8 graduate students (second trial).

We characterized the subjects by their experience and knowledge on four areas: agile software development, RE, software security and inspections. As a result, we found that the majority of students had a low level of security experience and knowledge. We also found that participants are not familiar with requirements inspection. Hence, they match our intended profile. Aiming at mitigating threats to validity concerning the distribution of subjects between groups, we used the characterization and applied the principles of balancing, blocking and random assignment [37]. Hence, students who demonstrated knowledge on software security (4 out of 33) were separated and distributed equally into the control and the experimental groups of each trial.

Subjects who found less than 10% of the defects were discarded as outliers, because, in our understanding, their results reflect lack of interest in having a good performance in the review. We discarded 5 subjects from the first trial. In the second trial, it was not necessary to discard subjects.

F. Experiment Design

Our experiment is composed of one factor with two treatments: (1) using our proposed reading technique and (2) directly using the OWASP high-level SRs and a list of defect types to be found. 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.[3] Figure 2 shows all the phases of the experiment.

[3]http://doi.org/10.5281/zenodo.3273298

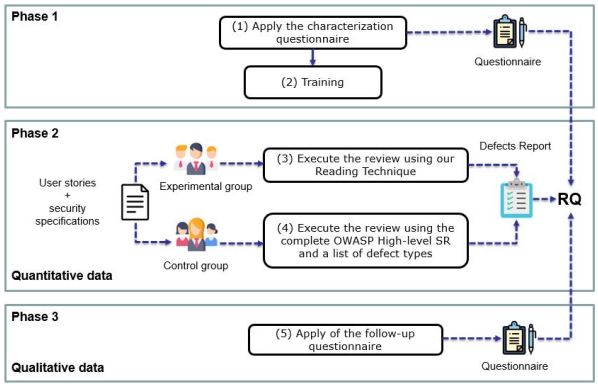

Figure 2. Experimental design

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 two trials. The students of each trial 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 experiment gave us feedback on the execution of the experiment. The instruments used to conduct the experiment are further described in the following.

G. Instrumentation

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

Follow-up questionnaire. This questionnaire was based on the Technology Acceptance Model (TAM) with 5-point scale. TAM has been extensively used in several studies [34]. 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.

Training. The training was focused on the security properties, the OWASP high-level SRs and on the defect types.

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, the technique was generated according to the user story and its related high-level SRs. For the other treatment the list of the security properties and the high-level SRs was provided together with the list of the defect types.

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 VIII.

H. Experiment Operation

The experiment was executed along two days. On the first day, the subjects answered the characterization form in order to allow dividing them into experiment groups. On the second day prior to the execution of the experiment, concepts of the security properties, high-level SRs, and defect types were reviewed by subjects in a training session. After that, the inspection was conducted as follows.

All subjects had up to one hour to finish the review. During the experiment, the control group used the OWASP high-level SRs and a list of defect types as support during the review. The experimental group used our approach. When the subjects from both groups finished the task, they had to fill out the follow-up questionnaire. The first and second trial were conducted with the undergraduate and graduate students, respectively.

I. Threats to Validity

Internal validity. First, aiming to avoid personal bias, we used researcher triangulation to collect and analyze all data. Second, we characterized all the subjects. The characterization allowed us to apply the blocking principle, which consisted of removing confounding factors by distributing the participants so that these characteristics were equally distributed among the groups. Random assignment was employed for subjects with similar characteristics. Finally, participants received the same training.

Construct validity. For our quantitative analysis, we used metrics (effectiveness and efficiency) that are commonly used in inspection experiments. Regarding the qualitative analysis, we used the TAM, which has also been widely used and evaluated [34].

Conclusion validity. This validity is related to the sample size and the statistical methods used. The statistical hypothesis testing methods were chosen according to the sample distribution.

External validity. As we planned to conduct a limited amount of trials with a limited amount of subjects, the experiment package is available for external replications. 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 [13]. Regarding the objects, we peer-reviewed the requirements specifications and the seeded defects in terms of their representativeness.

V. Results

In the following, we present the results of the experiment. We also describe the data collection and analysis procedure.

We executed three steps to collect the data necessary for answering our research questions. First, we collected the number of defects found. Based on this data, we evaluate the performance of the treatments in terms of effectiveness. We also collected the time spent for detecting defects to

compare the performance in terms of efficiency. Finally, we collected opinions of the subjects to receive feedback on the usefulness and ease of use of the approach.

Regarding the analysis procedure, we conducted both quantitative (RQ1) and qualitative analyses (RQ2). Related to the quantitative analysis, descriptive statistics were based on the metrics above. Thereafter, to answer RQ1, statistical hypothesis testing was applied. Our analysis was conducted using the statistical tool RStudio version 1.1.4. For hypotheses testing, we used the Mann-Whitney test with $alpha$ = 0.05. This choice of statistical significance and test was motivated by the small number of independent samples. For the qualitative analysis, we present the frequencies of responses to the TAM questionnaire. Additionally, grounded theory coding activities were applied to answer RQ2.

RQ1: Does the approach have an effect on defect detection effectiveness and efficiency?

We wanted to understand the potential of the approach to detect security defects in agile specifications of web applications. For this, we compared the performance of using our approach versus the review based on the OWASP high-level SRs and a list of defect types. Regarding the effectiveness, Figure 3 shows the distribution of the number of defects found by the students in each trial.

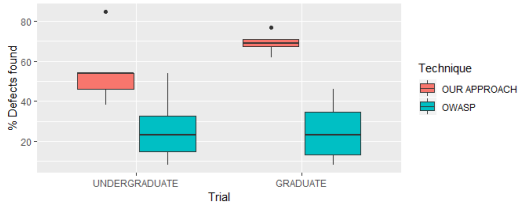

Figure 3. Defect detection effectiveness

Observe that in both trials our approach was more effective than the other review. In the first trial, the experimental group identified, in median, 54% of the defects while the control group identified 23%. The difference was higher when observing the performance of the second trial. 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 may have happened because we slightly modified the defect reporting form in the second trial to ease understanding and fulfillment. The reason was that several inspectors mentioned in the first trial that the defect reporting form was confusing. The change consisted of merge the column 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 capturing the same information.

We also wanted to test our null hypothesis on the effectiveness (H0a), i.e., we checked whether these differences were significant. The results of the tests allowed to reject the null hypothesis for both trials (p-values of 0.002 and 0.012 for the first and second trial, respectively); this means there is a significant difference in terms of effectiveness between our approach and the other defect-based technique. In addition, we calculated the Cohen's effect size [35] for both trials (3.46 and 2.24 for graduate and undergraduate students, respectively). Thus, we can partially answer RQ1: Our approach has a positive impact on defect detection effectiveness with a very large effect size.

After knowing the effectiveness of our approach, we can question its efficiency by analyzing the defects found per hour by the inspectors. Figure 4 shows the distribution of the efficiency of the subjects involved in the experiment.

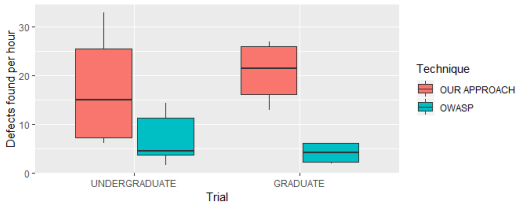

Figure 4. Defect detection efficiency

Note that the efficiency follows the same pattern of the effectiveness, that is, the number of defects found per hour by the experimental group was greater than the one of the control group. In the first trial, the median of our approach efficiency was 15 defects found per hour (we seeded only 13 defects, but participants took less than one hour to complete their tasks), while the median of the other one was four. In the second trial, the median of our approach efficiency increased to 21 defects found per hour versus four defects found per hour by the control group, i.e., inspectors who used our approach identified defects faster.

Regarding the statistical hypothesis testing for efficiency, we found that the Mann-Whitney Test suggests rejecting our second null hypothesis (p-values of 0.02 and 0.01 for the first and second trial, respectively). This means there is a significant difference in terms of efficiency between our approach and the OWASP review. Additionally, the relevance of this difference (Cohen's effect size [35]) was large for both trials (1.56 and 3.29 for undergraduate students and graduate students 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 the requirements defect types as a basis for verification.

RQ2: How do the inspectors perceive the usefulness and ease of use of the approach?

After inspectors reviewed the security specifications, we asked whether they found the approach useful and easy to use. Through the TAM questionnaire, we wanted to know about their perceptions on using our approach. All those who used our approach strongly agreed (75%) or partially

agreed (25%) that their performance improved in some way (find defects faster). This last one perception may be strongly related to the lack of experience and security knowledge of the inspectors and difficulties faced by them when conducting the review. For instance, one inspector stated the following: "The review may be exhausting and time-consuming because the task description document is not lightweight". The inspectors also mentioned some difficulties faced such as "confusing defect reporting form" and "requirement specifications are too abstract".

Regarding ease of use, again all those who used our approach strongly agreed (17%) or partially agreed (83%). The large number of partial agreements indicates that improvements could be added to facilitate the use of the approach. According to the follow-up questionnaire, the inspectors proposed some points that may improve the understanding 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 is convincing because some inspectors reported difficulties to adopt the approach. In contrast, three undergraduate students mentioned that once the review process using the approach is understood, the detection of defects is simple.

VI. Discussion

In the following, we discuss several further questions that have implications on future research.

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.

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.

Acceptance of the approach by practitioners. We saw that in principle our approach supports novice inspectors to detect defects related to security in agile specifications. We consider that this contributes to narrowing the security knowledge gap that exists between experts and novice inspectors. In addition, we designed the approach in such a way that it could work without expensive review cycles, aligned with the agile philosophy.

VII. Limitations

We concentrated on a set of concrete 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 are also aware that our security specifications 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.

VIII. Concluding Remarks

We proposed an approach for reviewing security-related aspects in agile requirements specifications of web applications. The approach considers user stories and security specifications as inputs and involves applying NLP in order to relate those user stories to security properties and OWASP high-level SRs. As a result, the approach provides a focused reading technique that can be used to support the manual inspection of agile specifications.

We validated the approach by designing and conducting two trials of a controlled experiment. The purpose was to validate the feasibility of using our approach for detecting defects related to security in web applications. In these trials, our approach had a positive effect on defect detection effectiveness and efficiency. Inspectors who used our approach identified more defects in less time than inspectors who conducted the inspection using the OWASP high-level SRs and a list of defect types. In principle, subjects found our approach useful and easy to use.

Future work includes evaluating the performance of using our approach in industry settings and when compared to other inspection techniques (e.g., PBR-security [7], even though it was not designed for the agile context). Furthermore, we consider it important to follow an example process integration, e.g., by further automatizing the approach within the FESRAS framework, in such a way that applying the reading technique could be guided by the framework. This could help mitigating the difficulties mentioned by the participants of the experiment. In addition, we are aware that our repository of keywords is still limited and that our approach could be evolved beyond exact keyword matching.

Acknowledgment

We would like to thank the CAPES agency for financial support. We are also grateful to all the experiment participants.

REFERENCES

[1] R. Araujo and M. Curphey, "Software security code review: Code inspection finds problems," *Software Magazine*, 2005.

[2] V. Basili, G. Caldiera, F. Lanubile, and F. Shull, "Studies on reading techniques," in *Proc. of the Twenty-First Annual Software Engineering Workshop*, vol. 96. Citeseer, 1996, p. 002.

[3] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sørumgård, and M. V. Zelkowitz, "The empirical investigation of perspective-based reading," *Empirical Software Engineering*, vol. 1, no. 2, pp. 133–164, 1996.

[4] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, "Manifesto for agile software development," 2001.

[5] B. Boehm, "Get ready for agile methods, with care," *Computer*, no. 1, pp. 64–69, 2002.

[6] L. Cao and B. Ramesh, "Agile requirements engineering practices: An empirical study," *IEEE software*, vol. 25, no. 1, pp. 60–67, 2008.

[7] J. C. Carver, F. Shull, and I. Rus, "Finding and fixing problems early: A perspective-based approach to requirements and design inspections," *STSC CrossTalk*, 2006.

[8] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Springer Science & Business Media, 2012, vol. 5.

[9] P. T. Devanbu and S. Stubblebine, "Software engineering for security: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*. ACM, 2000, pp. 227–239.

[10] D. Domah and F. J. Mitropoulos, "The nerv methodology: A lightweight process for addressing non-functional requirements in agile software development," in *SoutheastCon 2015*. IEEE, 2015, pp. 1–7.

[11] A. Eberlein and J. Leite, "Agile requirements definition: A view from requirements engineering," in *Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE02)*, 2002, pp. 4–8.

[12] F. Elberzhager, A. Klaus, and M. Jawurek, "Software inspections using guided checklists to ensure security goals," in *2009 International Conference on Availability, Reliability and Security*. IEEE, 2009, pp. 853–858.

[13] D. Falessi, N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka, and M. Oivo, "Empirical software engineering experts on the use of students and professionals in experiments," *Empirical Software Engineering*, vol. 23, no. 1, pp. 452–489, 2018.

[14] D. M. Fernández, S. Wagner, M. Kalinowski, A. Schekelmann, A. Tuzcu, T. Conte, R. Spinola, and R. Prikladnicki, "Naming the pain in requirements engineering: comparing practices in brazil and germany," *IEEE Software*, vol. 32, no. 5, pp. 16–23, 2015.

[15] P. Fusaro, F. Lanubile, and G. Visaggio, "A replicated experiment to assess requirements inspection techniques," *Empirical Software Engineering*, vol. 2, no. 1, pp. 39–57, 1997.

[16] K. M. Goertzel, T. Winograd, H. L. McKinley, L. J. Oh, M. Colon, T. McGibbon, E. Fedchak, and R. Vienneau, "Software security assurance: a state-of-art report (sar)," Information Assurance Technology Analysis Center (IATAC), Tech. Rep., 2007.

[17] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh, "Security requirements engineering: A framework for representation and analysis," *IEEE Transactions on Software Engineering*, vol. 34, no. 1, pp. 133–153, 2008.

[18] S. H. Houmb, S. Islam, E. Knauss, J. Jürjens, and K. Schneider, "Eliciting security requirements and tracing them to design: an integration of common criteria, heuristics, and umlsec," *Requirements Engineering*, vol. 15, no. 1, pp. 63–93, 2010.

[19] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in human behavior*, vol. 51, pp. 915–929, 2015.

[20] M. Kuhrmann, P. Diebold, J. Münch, P. Tell, V. Garousi, M. Felderer, K. Trektere, F. McCaffery, O. Linssen, E. Hanser *et al.*, "Hybrid software and system development in practice: waterfall, scrum, and beyond," in *Proceedings of the 2017 International Conference on Software and System Process*. ACM, 2017, pp. 30–39.

[21] G. Lami, S. Gnesi, F. Fabbrini, M. Fusani, and G. Trentanni, "An automatic tool for the analysis of natural language requirements," *Informe técnico, CNR Information Science and Technology Institute, Pisa, Italia, Setiembre*, 2004.

[22] G. McGraw, *Software security: building security in*. Addison-Wesley Professional, 2006, vol. 1.

[23] D. Mellado, C. Blanco, L. E. Sánchez, and E. Fernández-Medina, "A systematic review of security requirements engineering," *Computer Standards & Interfaces*, vol. 32, no. 4, pp. 153–165, 2010.

[24] D. Méndez Fernández, S. Wagner, M. Kalinowski, M. Felderer, P. Mafra, A. Vetrò, T. Conte, M.-T. Christiansson, D. Greer, C. Lassenius *et al.*, "Naming the pain in requirements engineering: contemporary problems, causes, and effects in practice," 2016.

[25] S. Nerur, R. Mahapatra, and G. Mangalaraj, "Challenges of migrating to agile methodologies," *Communications of the ACM*, vol. 48, no. 5, pp. 72–78, 2005.

[26] OWASP, *The Open Web Application Security Project, February*, 2019.

[27] F. Paetsch, A. Eberlein, and F. Maurer, "Requirements engineering and agile software development," in *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.* IEEE, 2003, pp. 308–313.

[28] B. Ramesh, L. Cao, and R. Baskerville, "Agile requirements engineering practices and challenges: an empirical study," *Information Systems Journal*, vol. 20, no. 5, pp. 449–480, 2010.

[29] E.-M. Schön, J. Thomaschewski, and M. J. Escalona, "Agile requirements engineering: A systematic literature review," *Computer Standards & Interfaces*, vol. 49, pp. 79–91, 2017.

[30] F. J. Shull and V. R. Basili, "Developing techniques for using software documents: a series of empirical studies," Ph.D. dissertation, research directed by Dept. of Computer Science.University of Maryland , 1998.

[31] J. Slankas and L. Williams, "Automated extraction of non-functional requirements in available documentation," in *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*. IEEE, 2013, pp. 9–16.

[32] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of network and computer applications*, vol. 34, no. 1, pp. 1–11, 2011.

[33] G. Travassos, F. Shull, M. Fredericks, and V. R. Basili, "Detecting defects in object-oriented designs: using reading techniques to increase software quality," in *ACM Sigplan Notices*, vol. 34, no. 10. ACM, 1999, pp. 47–56.

[34] M. Turner, B. Kitchenham, P. Brereton, S. Charters, and D. Budgen, "Does the technology acceptance model predict actual use? a systematic literature review," *Information and software technology*, vol. 52, no. 5, pp. 463–479, 2010.

[35] C. W. VanVoorhis and B. L. Morgan, "Understanding power and rules of thumb for determining sample sizes," *Tutorials in quantitative methods for psychology*, vol. 3, no. 2, pp. 43–50, 2007.

[36] H. Villamizar, M. Kalinowski, M. Viana, and D. M. Fernández, "A systematic mapping study on security in agile requirements engineering," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018, pp. 454–461.

[37] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[38] Z. Zhang, V. Basili, and B. Shneiderman, "An empirical study of perspective-based usability inspection," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 42, no. 19. SAGE Publications Sage CA: Los Angeles, CA, 1998, pp. 1346–1350.

[39] D. Zubrow, "Software quality requirements and evaluation, the iso 25000 series," *Software Engineering Institute, Carnegie Mellon*, 2004.