A Perception of the Practice of Software Security and Performance Verification

Victor Vidigal Ribeiro
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
vidigal@cos.ufrj.br

Daniela Soares Cruzes
SINTEF DIGITAL
Trondheim, Norway
danielac@sintef.no

Guilherme Horta Travassos
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
ght@cos.ufrj.br

***Abstract*— Security and performance are critical non-functional requirements for software systems. Thus, it is crucial to include verification activities during software development to identify defects related to such requirements, avoiding their occurrence after release. Software verification, including testing and reviews, encompasses a set of activities that have a purpose of analyzing the software searching for defects. Security and performance verification are activities that look at defects related to these specific quality attributes. Few empirical studies have been focused on how is the state of the practice in security and performance verification. This paper presents the results of a case study performed in the context of Brazilian organizations aiming to characterize security and performance verification practices. Additionally, it provides a set of conjectures indicating recommendations to improve security and performance verification activities.**

Keywords—security verification, performance verification, security testing, performance testing, case study research (keywords)

I. INTRODUCTION

The popularization and massive use of software systems bring benefits to the modern life. However, their broad availability increases the concerns regarding some critical software quality dimensions that were not in focus for many years. Security [1] and performance [2] are examples of such dimensions. Security is relevant due to the presence of critical and sensitive information manipulated and stored by the software systems while performing their tasks. The information usually requires high confidence and different levels of classification, having entailed a growing interest in accessing it to get improper benefits [3][4]. Performance is relevant due to the never-ending limitation of computational resources [5].

Software development organizations usually include quality assurance activities throughout the software life-cycle to evaluate the software quality, preventing failures after the software releases. Software verification [6], including testing and reviews, encompasses a set of activities aiming to analyze the software looking for defects. Security and performance verification are activities that look for defects regarding these specific quality perspectives. Different verification practices and techniques can be used individually or combined, promoting specific benefits and challenges to the verification of security and performance [7][8][9][10].

Despite the existence of some security and performance verification techniques, the software systems still present many defects related to these quality properties. For instance, performance issues account for a significant fault category in specific domains (e.g., telecommunications) [11] and news reporting systems attacks are increasingly frequent [12]. Some explanations regarding this situation could be (1) the inefficiency of used verification techniques, (2) the software organizations are not adopting adequate verification techniques, or (3) there is a lack of evidence-based verification techniques because of the apparent disconnection between academy and industry into this context [13]. Furthermore, automated attack scripts, the abundance of attack information, and global interconnection make it easier attack systems than it was before [14].

Some works available in the technical literature aim to characterize the software verification state of the practice. Most of them generically characterize software verification [15][16][17], including a family of surveys [18][19][20][20][21] and studies regarding verification in the context of specific software categories, such as distributed and heterogeneous systems [22], android applications [23], and safety-critical systems [24]. However, as far as we could investigate, few studies specifically discuss the state of the practice of security verification and no study relates to performance verification.

Therefore, this work aims to provide and evidence on how organizations are performing security and performance verification activities and practices in their software projects. We present the results of three cases carried out in different Brazilian organizations. Additionally, it was possible to identify a set of conjectures related to security and performance verification based on the analysis of collected information. The conjectures serve as the basis for recommendations in security and performance verification, confirming findings presented in [25][26]. The results from this study will, also, allow to identify possible technology gaps and challenges and provide feedback to software researchers regarding the alignment of their research with real problems [27].

Cruzes et al. [25] report the results of a case study performed in four software development organizations in Austria and Norway. They were concerned about how security testing practices have been performed in agile context. Similar to our results, they concluded that static code analysis and penetration tests (pen tests) are the most activities usually performed to evaluate software security. Additionally, they provide some recommendations for practitioners and researchers towards software testing: the need for more precision on the definition of security needs, the improvement of developers' security knowledge, and the need for lightweight techniques suitable for practical use. From the same research group, Oyetoyan et al. [26] present the findings of an action

research highlighting the developers' perceptions on using security static analysis tools, also in Norway. Some of the findings are also similar to our results: the high effort to configure the tools, the high number of false positives, and unknown real tools' capability. There are coherence and complementarity between these published recommendations and ours, strengthen the confidence in the findings and final set of recommendations.

The next section presents some definitions; Section III describes the case study methodology. Section IV presents the results grouped by research questions, and the discussions are presented as a set of conjectures in Section V. The threats to validity are presented in section VI. Finally, Section VII presents the conclusions, highlighting essential findings.

II. BACKGROUND

Overall, this work follows the concepts defined by ISO-29119 [28]. However, it is important to contextualize some of these concepts due to the lack of consensus about their definitions in Software Engineering. Therefore, we highlight the most important concepts discussed in this work, intending to be concise and with no intention of conceptualizing all software verification area (Table I).

TABLE I. MAIN USED CONCEPTS

Concept	Interpretation
Verification practice	What is performed for supporting verification, e.g., unit testing.
Verification technique	How the verification is performed. E.g., boundary value to generate test cases
Definition of *done*, acceptance criteria, or stop criteria	Overlapping concepts. The definition of *done* is used as a criterion to conclude a verification activity.
Automation level	When looking at the automation level, we need to know if a practice is manually or automated performed.
Asset	The part of the system covered by the verification practice, e.g., the source code is an asset regarding static code analysis. It is not defined as an artifact because the verification may target the running system, which is not an artifact.
Vulnerability	Generally used to designate security-related issues. In this work, the term defect is used since it is more generic and can also be used to represent both security and performance issues.

III. METHODOLOGY

Table II shows the characterization of three different Brazilian organizations in which the study was performed. We believe that such different organizations profiles strengthen the possibility of generalizability of the results. The last column represents the type of method used for data collection. Observation means that the researcher gathers information through face-to-face monitoring of verification team activities. On Interview method, the researcher followed a set of pre-defined questions to gather information and, in the questionnaire, a set of printed questions were delivered to the respondents.

TABLE II. ORGANIZATIONS DESCRIPTION

ID	Nature	# Employees (# Developers)	# Subjects	Data Collection Method
Org1	Governmental	~10599 (unknown)	5	Observation, Interview, Questionnaire
Org2	University laboratory	~154 (~132)	2	Interview, Questionnaire
Org3	Private	~250 (~150)	2	Interview, Questionnaire

Org1 is a large governmental organization that provides information technology services to the Brazilian government with 10599 employees (most of them are developers). We performed observations, interviews, and questionnaires with a verification team composed of five employees. The university laboratory, identified as *Org2*, with about 154 employees (132 are developers), develops technical solutions to the Brazilian government, including the development of software. We performed interviews and questionnaires with two employees that are responsible for security and performance verification. The *Org3* is a private company with about 250 employees and 150 developers. The company develops credit card payment systems, and we gathered data from two employees through interviews and questionnaires.

A set of artifacts were used to support the study, including the case study protocol and a presentation letter aiming to ease the contact with the organizations' representatives (Table III).

TABLE III. CASE STUDY INSTRUMENTS DESCRIPTION

ID	Description	Objectives
P1	Case study protocol	The protocol followed by the case study.
C1	Presentation letter	A letter used to make the first contact with the organizations, characterizing the researchers involved and the objectives of the study.
I1	Organization agreement term	After the organization agrees with the research, its representative signs the agreement term. It marks the beginning of the case study.
I2	Participant agreement term	In the first contact with each participant, they must sign the consent term to allow us to collect and use data.
I3	Organization characterization	Data can be gathered from different sources as the participants and organization website. It was filled in during different moments of the case study execution.
I4	Project characterization	It supports data collection during participants interview and questionnaire.
I5	Participant characterization	It supports data collection after participants interview through a questionnaire.
I6	Verification practices identification	Data collected in different stages of case study execution. It was gathered from observation, interview, and questionnaire.
I7	Verification of decision-making factors	Data collected in different stages of case study execution. It was gathered from observation, interview, and questionnaire.

ID	Description	Objectives
I8	Participant opinion	It supports data collection after participants interview trhough questionnaire.

IDENTIFICATION OF SECURITY AND PERFORMANCE PRACTICES

Project ID _____

1. What are the practices used to perform security and performance verification?

A. Practice_____

B. Responsible _____

C. Technique _____

D. Asset _____

E. Tools _____

F. Description

_____

Fig. 1. Verification Practices Extraction Form (I6)

Additionally, a set of instruments was used to support the data collection. The most important instrument (Fig. 1 - I6) aims to answer the research questions directly; the other instruments aim to formalize the research agreement and characterization of organizations and participants. The first author of this paper filled the instruments when collecting data during observations and interviews. The participants filled out the questionnaire when it was used to collect data.

After collecting data from the three organizations, about 38 artifacts were filled out (instruments, interviews transcriptions, and observation notes). Then, the first author qualitatively analyzed them by following a coding process done by the first author of this paper. Additionally, the second and third authors iteratively revised the generated codes in several meetings sections throughout the process. The MAXQDA[1] tool was used to support the coding process into which all completed instruments were imported. 892 excerpts from artifacts were grouped in 775 codes.

Furthermore, during the coding process, it was possible to identify categories that do not directly support answering the research questions, but it could provide essential findings of security and performance verification, complementing the study results. Thus, such information was organized into a code category classified as Conjectures (Inference formed statements without proof or sufficient evidence [29]), representing recommendations aiming to improve the benefits of security and performance verification activities.

IV. RESULTS

This section describes the results of cases carried out in three different Brazilian organizations. We followed the research questions and the sub-research questions as follows:

RQ1: Which are the practices used by the organizations to support the verification of security and performance?

RQ 1.1. What are the standard techniques?

RQ 1.2. Which definition of done do they adopt?

RQ 1.3. How is the level of automation?

RQ 1.4. What are the assets covered?

The practices (RQ1) used in security and performance verification are presented in Fig. 2. The Fig. 3 and Fig. 4 present the identified techniques (RQ1.1), the definition of the *done* criteria (RQ1.2), automation level (RQ1.3), and assets (RQ1.4) related to each verification practices. The practices details are also described as follow, grouped by research questions.

A. RQ1: Software Security and Performance Verification Practices

As shown in Fig. 2, static code analysis is performed in two ways regarding security, either triggered by a tester analyst or embedded in a continuous integration tool (e.g., Jenkins). In the first case, the code inspection depends on human action, and it is the verification team's responsibility to perform this practice. In the second, the code inspections mandatorily happen when the programmer commit the code to the repository.

The penetration testing is performed at the end of the software development lifecycle. It is usually performed only for critical systems (or a part of them), and the product owner defines delivery as critical because an attack could harm the organization reputation. Security specialists or the verification team are who usually perform the penetration tests.

Regarding log inspection, we classify it as a verification practice when it is used to identify software failures and as a debugging practice when it is used to identify software faults (failures cause). Further, this classification became comfortable because specialists participating in the case study mentioned such activity as a verification practice. For instance, one interviewee said: "*In a project, I participated in, there was one IP accessing the system, trying to identify if it was built in PHP and Wordpress. So, it was a well-known vulnerability they were searching for*". Therefore, the log inspection allows the identification of what could be considered a security failure: the application was configured to show the technologies used.

The verification team performs response time test, resource consumption, and log inspection. Response time test, which is the execution of the software aiming to evaluate the amount of time from a request to a response, is performed at the end of a development iteration (e.g., a sprint). These practices are not used to assess all system scenarios, but the analysts (e.g., architecture, business) or the verification team selects some system scenarios to be assessed.

The resource consumption test is also performed at the end of a development iteration and uses the same test cases regarding response time tests. However, the system scenarios evaluated by this practice is a subset of scenarios evaluated by the response time test. Log inspection is also used as a practice to performance verification, and it is performed aiming to identify significant delays from a request to a response.

[1] https://www.maxqda.com/

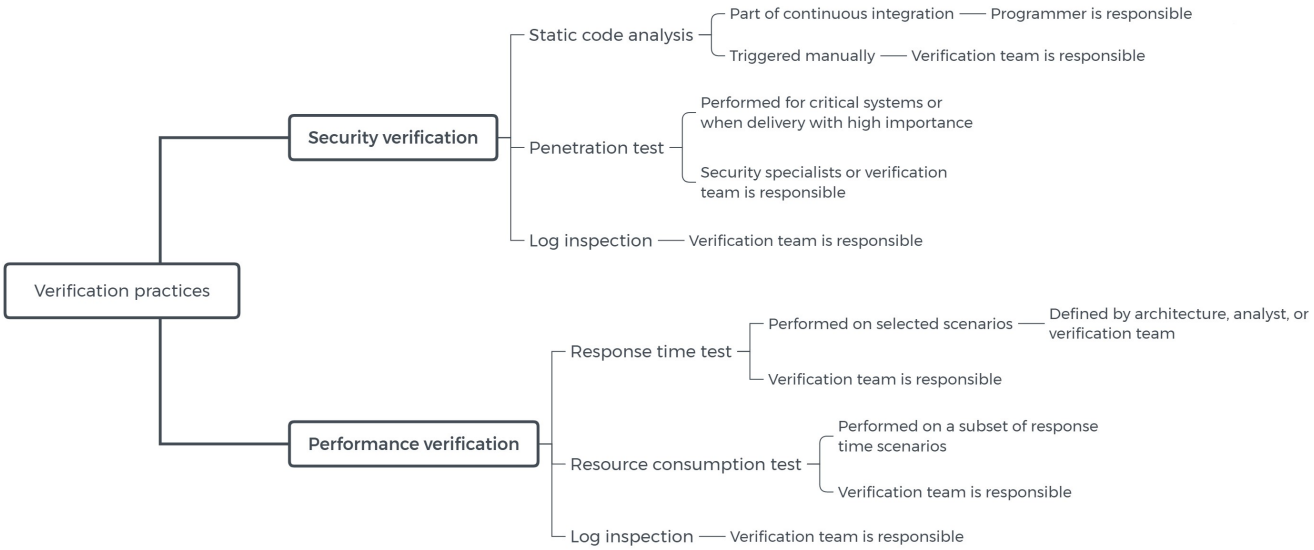

Fig. 2. Identified security and performance verification practices

B. RQ1.1: Software Security and Performance Techniques

The security and performance verification practices are executed through the use of verification techniques, providing a systematic way to build test cases or review procedures, and support for the definition of coverage and stop criterion. There were different categories of techniques used to evaluate security (Fig. 3): tool-based, failure-based, experience-base, and *ad-hoc*.

On the tool-based practices, the technique is embedded in the tool. Failure-based techniques make use of known failures (e.g., common vulnerabilities databases), generating test cases addressed to identify these faults. In experienced-based techniques, the verification team makes use of their own experience to generate test cases or perform inspections. When *ad-hoc*, the practice is performed in an aleatory and non-systematic way.

The Static Code Analysis is usually performed through a tool, meaning that a tool is executed, and the verification results are analyzed. An observed issue with the usage of this technique is that in many cases, the verification team is usually not aware of what the tool is verifying and what are the limitations of the tools regarding their fault detection capability.

The Penetration Test is usually performed through a failure-based technique, and the test cases are built aiming to explore known defects available on common security vulnerabilities repositories. Another technique used for the penetration test is experience-based in which a security specialist knowledge plays the role of a malicious user trying to access the system.

The Log Inspection can be considered an *ad-hoc* practice because the inspection is based on unknown criteria. A security specialist performs it.

Regarding performance practices (Fig. 4), both the Response Time Test and Resource Consumption Test use techniques based on the tester experience or based on similar systems. In the second case, the verification team verifies whether the current system can reuse test cases from previous systems of the same company. As for security, the performance Log Inspection is *ad-hoc,* and the verification team performs it.

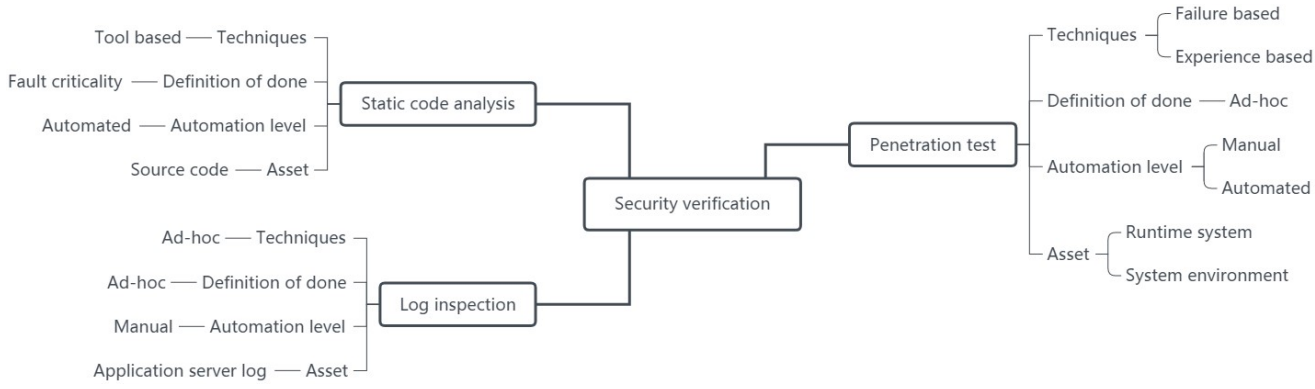

Fig. 3. Software Security Verification Practice Details

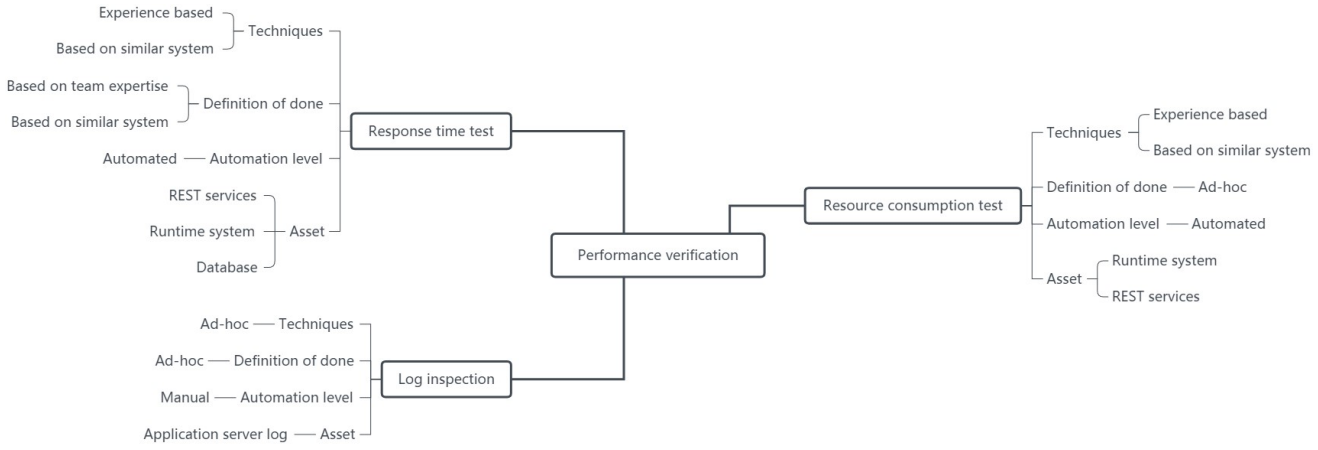

Fig. 4. Software Performance Verification Practices Details

C. *RQ1.2: Software Security and Performance Definition of Done Criteria*

The definition of *done* is the criterion used to define the conclusion of verification activities, signalizing that the software development can move to the next phase, usually delivery. We identified four categories to the definition of *done*, based on fault criticality, team-experience, similar system, and *ad-hoc*. On the definition of *done* based on fault criticality, the faults identified by verification are classified regarding their criticality level (e.g., low, medium, and high). Verification activities are defined as completed when no more defects of a defined criticality level is identified. The static code analysis makes use of this criterion. However, the minimum criticality level to define code analysis should re-run is not correctly defined. One of the participants said: "*The defects (vulnerabilities) reported by Fortify are classified, by the tool, according to the criticality level. So, in some situations, we used such criticality level to define what faults would be fixed. Thus, the most critical faults were fixed. However, there is no a constraint defining what criticality level define that faults must be fixed. Such a decision was a cross-team agreement*".

When based on team experience, the verification team meet and decide whether verification activities should be continued. Besides, the verification team may decide to look into a similar system to decide when the verification activities should end. For instance, in the last similar system, the tests were set as concluded after each test ran three times successfully. Thus, the current system tests stop after three tests battery. The response time makes use of these two kinds of definition of *done*.

If the definition of *done* is *ad-hoc*, there is no systematic way to set the verification as concluded and it is performed randomly. The penetration test, log inspection, and resource consumption use the *ad-hoc* definition of *done*.

D. *RQ1.3: Software Security and Performance Automation Level*

The evaluation of automation level classifies the verification practice between manually and automated[2].

Static Code Analysis is always automated, the most common tools in our case studies were Brakeman and HP Fortify SCA. Additionally, Jenkins, Sonar, and Threadfix are used as auxiliary tools to orchestrate the execution or to visualize results. When using static code analysis tools, participants reported that a problem is a large number of false positives.

The Penetration Test practice is performed manually (supported by scripts) or by using a support tool; the most common tools are Arachni, OWASP Zed Attack Proxy (ZAP), XSS ME, SQL Injection ME, Burp Suite, Meta Exploit, NMAP, and Whatweb.

The Response Time and Resource Consumption Tests are always automated and make use of similar tools, such as JMeter, Postman, BlazeMeter, Goldeneye, HTTPerf, SoapUI, CA APM, and tools developed by the own organization. There are also auxiliary tools such as Jenkins, spreadsheets.

The Log Inspection for both security and performance is manual. Probably, it indicates a lack of log inspection tools. One of the participants informed the performance failures identified by performing log inspection might have many false positives, because it is not easy to conclude if a detected anomaly is a regular user behavior or a system bottleneck: "*...when looking at logs, if I identify a 10 minutes delay from a request to another request, the problem is that I do not know if the request took 10 minutes to complete or if the user clicked on one option and then clicked on another only 10 minutes later*".

E. *RQ1.4: Software Security and Performance Assets*

The assets covered by security verification practices are source code, application server log, the system in execution,

[2] Links to website tools are available at http://lens.cos.ufrj.br/nfrwiki/index.php/Verification_tools

and the environment. Performance verification practices cover REST services, the system in execution, database, and application server log. It is important to notice that verification practices in the case studies do not cover early development phases artifacts.

V. DISCUSSION

We have produced a set of conjectures that can be seen as recommendations that must be considered during security and performance verification as a complementary of the results. Thus, we discuss the results grouped by these conjectures. The next sections present the nine conjectures.

A. Security and Performance Verification Requires a Suitable Environment.

A suitable environment is essential for verification. The execution of verification activities in a non-isolated environment is sometimes affected by external influences. Besides, inappropriate hardware/server configurations can generate a false perception of system behavior.

In the context of this study, the first conjecture arose from the observation that sometimes the security and performance verification share the same environment used by other activities. For example, in one organization the performance tests were executed on the same server used for system user acceptance test. In this case, there was a bidirectional influence. Thus, the performance tests may jeopardize the user acceptance activities, because the simulation of a large number of users operating the system causes hardware overloaded. On the other hand, when users were using the system for acceptance testing, the performance tests presented random results (e.g., aleatory response time), because it was not possible to know how the users were using the system.

The local network (or virtual private network) also influences the performance testing results. For instance, if the machine used to trigger performance tests make use of default organization network, the requests and responses may be delayed due overload of the network nodes (e.g., routers and sweets) that route them to the server the system is running.

Another issue regarding the verification environment is about the difference between the hardware configuration used for verification and used in production time. In some cases, the hardware used in the production environment is more powerful than the hardware used in verification activities, and this may result in a false result on system performance.

When facing unsuitable verification environment, the verification team tries to mitigate such issues by, for example, executing each test case more than once and at different moments, trying to mitigate external influences in the test results. One of the participants said: "*It is not possible to rely on the response time of only one scenario execution, because there may be interference that impairs the operation of the system. Thus, response time analysis only occurs after the scenario has been successfully executed three times*".

Neto et al. [30] mention the financial unfeasibility of using physical machines to compose the verification environment, while others works point to virtualization as the most appropriate technology for verification environments. However, some issues still need to be addressed for the practical use of virtualization technology: the number of supported virtual machines estimation, limit of the amount of virtual machine, test trigger response time is not stable, physical machine overload [31][32][33].

B. Security and Performance Verification Requires Suitable Techniques.

Performing the verification in an *ad-hoc* way hinders the definition of a criterion to select test cases and to have a definition of *done*. A proper verification technique naturally provides these criteria.

Such conjecture was coined due to a large number of practices conducted *ad-hoc*. For instance, the use of tools without precisely knowing what technique is implemented generates uncertainties about tool detection capability. Additionally, the number of false-positives defects identified by security tools can be a problem, if all the defects reported should be analyzed, it requires a substantial extra effort.

In our study, we could not find mitigation actions to these problems, but we noticed that the teams are aware that they can improve verification activities.

Martin and Xie [34] present the results of an experiment showing the use of a technique increase the defect detection capability and the essential coverage of security verification. Furthermore, the use of suitable techniques in different phases of the software development (e.g., abuse case in requirements and model, misuse cases, threat trees in design) promotes the identification of defects in early stages of software development [35][36][37][38]. Some researchers also suggest a combination of techniques to increase the ability to detect different types of defects, such as complementing the automated tests with manual reviews [38][39][40].

C. The lack of security and performance requirements prevents the verification from playing its original purpose.

The lack of security and performance requirements prevents the verification from playing its original purpose (i.e., assessing whether the software meets its requirements) because in the absence of an oracle it is impossible to know if the verification results are correct. Also, inaccurate requirements overload other teams (e.g., analysts, architects, developers) because the verification team must continuously contact them.

In the organizations used in this study, sometimes there were no written performance requirements that could be used as an oracle. In such cases, the verification activities were not performed to assess whether the software meets its requirements but are to evaluate the capacity of the system. Some other times, the verification activities were performed based on subjective or imprecise requirements. For example, a participant reported a case where the tests were performed based on a brief description about users' behavior: "*In this system, everyone comes in at 8 in the morning and stays until 10 o'clock. Then they leave the system and come back at lunch*".

In the literature, researchers have identified a set of issues and challenges regarding security and performance requirements: lack of support tools and techniques, techniques are unsuitable to target users, requirements not provided, and wrong requirements descriptions. Such issues make verification impossible, ambiguous or generic [41][42][43][44]. It is also possible to identify a set of proposed techniques and recommendations to handle security and performance requirements: misuse cases, SETAM UMLsec, abuse cases, description of attack patterns [35][41][44][45][46][47][48][49][50], indicating a gap between practice and academy because despite the existence of techniques they are not in use.

D. *Training in Security and Performance Verification Improves Verification Activities.*

Some participants reported the challenges in performing security and performance verification due to the lack of training. One of the participants said: "*...security and performance tests were performed by a specific specialized team. However, the responsibility for testing now belongs to the department. In this way, we still feel a bit of difficulty in performing security and performance verification*".

The lack of training also results in a misunderstanding of security and performance verification concepts, generating risks to the verification activities, because team members may understand and perform their activities in a contradictory way. One way to mitigate the lack of consensus about verification concepts was to define a document precisely describing each verification activity and its purpose. A participant said: "*...usually to perform these tests I have to create a document of concepts definitions because there were several misunderstandings before. There are people who think that load testing means to verify the limit of the application and there are people who think that it is the stress testing*".

Røstad et al. [51] alert that few universities teach how to build secure software, resulting in students graduating without seeing safe development. They raise some critical issues regarding security training, such as ethics needs to be considered and is an educator's responsibility to teach the importance of dealing with security in the early phases of software development; they also mention the limited capability of open sources tools and a high cost of suitable tools. Bondi and Ros [52] highlight some issues regarding remote training of performance verification, such as the need to be systematic, problems with different time zone, and inculcate good performance testing practices.

E. *Verification Activities Only do Not Guarantee Software Security and Performance Effectiveness.*

Security and performance verification should be corroborated with other techniques because they alone do not create enough evidence to provide security confidence and adequate performance to software systems. In our study, participants of different organizations emphasized that verification activities not be entirely effective regarding defect detection capability. Therefore, it is vital to make use of complementary activities to improve software security and performance, such as system monitoring and actions to prevent the use of social engineering. A participant said: "*...there is a security monitoring team looking at the production servers. When an attack is identified, this team identify the exploited vulnerability and notify the development team who provides the fix*".

This finding is consistent with the software testing theory that states that testing aims to shows presence of defects, but it is not able to prove their absence [53]. Additionally, there is evidence in the literature that verification security methods do not identify all defects [54][55] and some performance tools has limitations such as on temporal synchronization, GUI elements identification and incomplete implementations [56].

F. *Security and Performance Verification Requires Cross-Functional Teams.*

Verification activities are not performed in isolation by only one team. It requires interaction between different teams, including different skills. In the case studies, there was a need for infrastructure, database, and technical teams to work together due to the need to configure servers (e.g., allow a specific IP to access the server or restart it after catastrophic failure), restore database backups and deal with specific low-level technologies. A participant reported a need to interact with an operating systems specialist: "*...we have to ask to change the (operating system) kernel because it has a limit in the size (of requests) that can be sent...*".

A cross-team interaction is also essential to identify the technologies used to build the software and how such technologies may influence verification results. For example, the verification team observes that performance tests of the same scenario were resulting in a shortened response time. Thus, talking to the development team, they discover the system was making use of a CDN (Content Delivery Network) cache technology.

Some papers identify the need for teams with different skills in security and performance verification activities and propose strategies to encourage the information exchange between teams [40][57][58]. A card game named protection poker provides security knowledge sharing, involves the entire development team, and increase the awareness of software security needs. Another recommendation is to take programmers as allies, not as enemies. Regarding performance, Johnson et al. [58] show how weekly meetings involving performance architect, domain experts, marketing stakeholders and developers can improve team interactions.

G. *Planning and Environment Configuration of Security and Performance Verification Demands Extra Effort.*

In the studied organizations, many of the verification activities are automated. Thus, the execution phase does not require as much effort as the planning and environment configuration. The planning phase demands high effort because the team should prioritize the test cases to be executed and identify dependencies between scenarios being tested. It requires an information exchange between verification and development teams.

The configuration of the environment also requires extra effort. A participant reported that the activity requiring the

most effort is to config the environment so that the system runs in a suitable state for testing: "*The major problem with performance testing is to organize the database with a suitable dataset, aiming to run the system under the desired load*.". In our study, we cannot find ways to reduce the effort of planning verification and environment configuration.

Chen et al. [59] present the challenges to configure an environment: be able to simulate workloads similar to the real world, to consider high insertion speed, and to meet BD and business logic constraints.

H. Organizational Support Influences Positively Security and Performance Activity.

This conjecture is about how the organizational perception of the importance of security and performance software systems properties affects the verification activities. Participants reported that organizations generally do not give proper attention to system security and performance, considering security and performance verification a waste of resources. Usually, organizations are inclined to worry about the security and performance of their systems only after they have a problem. Another situation where organizations invest more in security and performance is before a major release in which a security or performance failure could negatively affect organization image. One of the participants said: "*By my own experience working in different places, what I see is that people only care about testing when a problem happens.*".

Horký et al. [60] report an experiment showing that keeping programmers well-informed about performance can decrease the number of bad decisions, influencing the system performance. Besides, Ferrell et al. [61] emphasize the challenge of security awareness: programmers are not concern about security because they have a false impression that new development technologies are immune to security problems.

I. Security and Performance Verification Activities are Ignored When There Are Limited Resources of Time and Budget.

Security and performance verification activities are ignored when there is a reduction in the deadline or budget of the development project. It can impact the system quality because the product owner may decide to stop the verification before all activities are executed, decreasing verification coverage.

A participant informed the planned tests are reported in the final testing report, including the non-performed tests to mitigate it. Thus, the customers are aware that the executed tests did not the planned coverage, transferring the responsibility of the risk to the customer.

We were not able to find support for this conjecture in the technical literature.

VI. THREATS TO VALIDITY

We describe the threats to validity following the recommendations of Cruzes and Othmane [62] and making use of quality criteria (Q1-4) and proposed methods (M1-6) to improve them [63][64].

The credibility (Q1), representing the quality of being convincing or believable, was addressed using rich data/persistent observations (M1) and through data collection using three methods (observation, interviews, and questionnaires), making notes about what happened, and verbatim transcripts of what participants said. Furthermore, while reporting the results, we provide quotes of the participants.

The transferability (Q2) quality refers to the degree to which the results can be generalized to other contexts or settings. Such quality is problematic in the case of studies because it is not viable to achieve a significant number of subjects as it was not in our case. However, to improve the transferability, we used the intensive long-term involvement (M2) method, performing the research in place, making it possible to have a more precise context perception the study is characterizing. Thus, it was possible to provide an in-depth description of the organizations' characteristics and context in which data were collected.

Regarding dependability (Q3), the data stability and reliability over time and conditions, we performed the study in different organizations with participants from a variety of profiles. Furthermore, one more case was planned, and it is already scheduled to be performed. In this way, we can triangulate (M3) the results, improving the dependability. Additionally, the research protocol is available, making it possible the study replication in different contexts.

To avoid researcher's bias and improve the confirmability (Q4), we use peer debriefing (M4), exposing our main findings to a research group and discussing the coherence of them. We also promoted iterative meetings among the authors to discuss the codes according to creation progress. Additionally, we performed a search on the literature to support the coined conjectures.

It is important to mention what mitigated regarding threats to validity: due to participants availability restrictions we could not use respondent validation (M5) and member checking (M6) to confirm what they said and the validity of our conclusions.

Besides, it is important to mention that the case study was carried out in the context of Brazilian organizations, where Portuguese is the mother language. Thus, the participants' quotes reproduced here are translations of what was said by them. Besides, the artifacts and codes were built in Portuguese and translated into English, for the sake of readers' understanding. We believe that the translation does not affect the results reported since we did not perform a sentiment/feelings analysis of the participants' answers.

Finally, investigating two non-functional requirements together can be risky. There were situations in which it was not possible to determine whether a respondent was reporting issues related to security or performance.

VII. CONCLUSION

This paper presents the results of a case study performed in the context of three Brazilian software organizations. The study provides a characterization of the state of the practice of security and performance verification activities. Additionally,

the findings are discussed in the form of conjectures, representing recommendations applicable to such activities. In general, there is an increasing awareness of the importance of security and performance of software systems, and consequently, the importance of verification activities. However, there is a lack of knowledge about how verification should be accomplished.

The security and performance verification was performed on an unsuitable environment. Besides, the use of verification techniques is low. For instance, techniques aiming to systematize the test case generation or inspection procedure. It may result in inefficient test cases (low chance to reveal a failure). Such warning is emphasized by the profile of the professionals who perform the verification. Usually, they do not have suitable experience and training regarding security or performance. In addition, we conjecture that the lack of requirements or their imprecise specification contributes to the *ad-hoc* definition of *done*.

Finally, it was not possible to identify verification activities addressed to artifacts related to early stages of the software development cycle (e.g., requirements and design diagrams). It contradicts the recommendations of established guidelines [65][66].

As future work, we intend to perform a set of rapid reviews [66] to increase the confidence regarding the presented conjectures. Additionally, we started the replication of the case study in another organization, aiming to provide cross-validation of the current results.

We believe that the recommendations presented in this paper can help practitioners to avoid known problems conducting security and performance verification. Besides, it can be useful for researchers as they may target their investigations to address the raised issues. As future work, we intend to propose an approach to help practitioners perform the security and performance verification considering the conjectures as recommendations.

Acknowledgment

We much appreciate the support of participating organizations, and CAPES. Prof. Travassos is a CNPq Researcher. This work was partially supported by the SoS-Agile project: Science of Security in Agile Software Development, funded by the Research Council of Norway (grant number 247678).

REFERENCES

[1] D. Ameller, M. Galster, P. Avgeriou, and X. Franch, "A survey on quality attributes in service-based systems," *Software Quality Journal*, 2015.

[2] A. Caracciolo, M. F. Lungu, and O. Nierstrasz, "How Do Software Architects Specify and Validate Quality Requirements?" European Conference on Software Arqchitecture, 2014

[3] M. Labs, "McAfee Threat Predictions," 2016.

[4] I. B. M. X. Threat and I. Index, "IBM X-Force Threat Intelligence Index," 2017.

[5] H. S. Zhu, C. Lin, and Y. D. Liu, "A Programming Model for Sustainable Software," *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, pp. 767–777, 2015.

[6] IEEE-610.12, "IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990). Los Alamitos," *CA: IEEE Computer Society*, vol. 121990, 1990.

[7] M. Atifi, A. Mamouni, and A. Marzak, "A Comparative Study of Software Testing Techniques," *International Conference on Networked Systems*, 2017, pp. 373–390.

[8] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, "Security Testing: A Survey," *Advances in Computers*, vol. 101, pp. 1–51, 2016. https://doi.org/10.1016/bs.adcom.2015.11.003

[9] J. A. Meira, E. C. de Almeida, D. Kim, E. R. L. Filho, and Y. Le Traon, "'Overloaded!' — A Model-Based Approach to Database Stress Testing," *International Conference on Database and Expert Systems Applications*, 2016, pp. 207–222.

[10] Microsoft, "Performance Testing Guidance for Web Applications." [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/bb924375%28v%3Dpandp.10%29. [Accessed: 19-Jul-2018].

[11] A. Bertolino, "Software Testing Research : Achievements , Challenges , Dreams Software Testing Research : Achievements , Challenges , Dreams," *IEEE Transactions on Software Engineering*, no. September, pp. 85–103, 2007.

[12] Symantec, "Symantec Internet Security Threat Report," 2017.

[13] V. Garousi and M. Felderer, "Living in two different worlds: A comparison of industry and academic focus areas in software testing," *IEEE Software*, 2017.

[14] R. B. Vaughn, R. Henning and K. Fox, "An Empirical Study Of Industrial Security-Engineering Practices" *Journal of Systems and Software*, 21002, pp. 225–232. https://doi.org/10.1016/S0164-1212(01)00150-9.

[15] A. Causevic, D. Sundmark, and S. Punnekkat, "An Industrial Survey on Contemporary Aspects of Software Testing," *Third International Conference on Software Testing, Verification and Validation*, 2010, pp. 393–401.

[16] V. Blondeau, A. Etien, N. Anquetil, S. Cresson, P. Croisy, and S. Ducasse, "What are the Testing Habits of Developers? A Case Study in a Large IT Company," *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 58–68.

[17] B. Maqbool, F. U. Rehman, M. Abbas, and S. Rehman, "Implementation of Software Testing Practices in Pakistan's Software Industry," *International Conference on Management Engineering, Software Engineering and Service Sciences*, 2018, pp. 147–152.

[18] V. Garousi and J. Zhi, "A survey of software testing practices in Canada," *Journal of Systems and Software*, vol. 86, no. 5, pp. 1354–1376, May 2013.

[19] V. Garousi, A. Coşkunçay, A. Betin-Can, and O. Demirörs, "A survey of software engineering practices in Turkey," *Journal of Systems and Software*, vol. 108, 2015, pp. 148–177.

[20] A. C. Dias-Neto, S. Matalonga, M. Solari, G. Robiolo, and G. H. Travassos, "Toward the characterization of software testing practices in South America: looking at Brazil and Uruguay," *Software Quality Journal*, vol. 25, no. 4, 2017, pp. 1145–1183.

[21] S. M., M. Shamsur, A. Z., and M. Hasibul, "A Survey of Software Quality Assurance and Testing Practices and Challenges in Bangladesh," *International Journal of Computer Applications*, vol. 180, no. 39, 2018, pp. 1-8.

[22] B. Lima and J. P. Faria, "A Survey on Testing Distributed and Heterogeneous Systems: The State of the Practice," *International Conference on Software Technologies*, 2017, pp. 88–107.

[23] M. Linares-Vasquez, C. Bernal-Cardenas, K. Moran, and D. Poshyvanyk, "How do Developers Test Android Applications?," *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 613–622.

[24] M. Kassab, "Testing Practices of Software in Safety Critical Systems: Industrial Survey," *20th International Conference on Enterprise Information Systems*, 2018, pp. 359–367.

[25] D. S. Cruzes, M. Felderer, T. D. Oyetoyan, M. Gander, and I. Pekaric, "How is Security Testing Done in Agile Teams? A Cross-Case Analysis of Four Software Teams," *International Conference on Agile Software Development,* 2017, pp. 201–216.

[26] T. D. Oyetoyan, B. Milosheska, M. Grini, and D. Soares Cruzes, “Myths and Facts About Static Application Security Testing Tools: An Action Research at Telenor Digital,” *International Conference on Agile Software Development*, 2018, pp. 86–103.

[27] V. Garousi, M. Felderer, M. Kuhrmann, and K. Herkiloğlu, “What industry wants from academia in software testing?,” *21st International Conference on Evaluation and Assessment in Software Engineering - EASE*, 2017, pp. 65–69.

[28] ISO 29119-1, “Software and systems engineering - Software testing - Part 1: Concepts and definitions,” vol. 2013, 2013.

[29] “Conjecture,” “Merriam-Webster.com,” 2011. [Online]. Available: https://www.merriam-webster.com. [Accessed: 16-Jul-2018].

[30] M. A. S. Netto, S. Menon, H. V. Vieira, L. T. Costa, F. M. de Oliveira, R. Saad, and A. Zorzo, “Evaluating Load Generation in Virtualized Environments for Software Performance Testing,” *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, 2011, pp. 993–1000.

[31] M. M. Arif, W. Shang, and E. Shihab, “Empirical study on the discrepancy between performance testing results from virtual and physical environments,” *Empirical Software Engineering*, vol. 23, no. 3, pp. 1490–1518, Jun. 2018.

[32] G.-H. Kim, Y.-G. Kim, and K.-Y. Chung, “Towards virtualized and automated software performance test architecture,” *Multimedia Tools and Applications*, vol. 74, no. 20, pp. 8745–8759, Oct. 2015.

[33] S. Gaisbauer, J. Kirschnick, N. Edwards, and J. Rolia, “VATS: Virtualized-Aware Automated Test Service,” *Fifth International Conference on Quantitative Evaluation of Systems*, 2008, pp. 93–102.

[34] E. Martin and T. Xie, “Automated Test Generation for Access Control Policies via Change-Impact Analysis,” *Third International Workshop on Software Engineering for Secure Systems*, 2007, pp. 5-5.

[35] J. McDermott and C. Fox, “Using abuse case models for security requirements analysis,” 15th Annual Computer Security Applications Conference (ACSAC’99), 1999, pp. 55–64.

[36] I. Alexander, “Misuse cases: use cases with hostile intent,” *IEEE Software*, 2003, pp. 58–66.

[37] A. Marback, H. Do, K. He, S. Kondamarri, and D. Xu, “Security test generation using threat trees,” *ICSE Workshop on Automation of Software Test, 2009*, pp. 62–69, 2009.

[38] H. Omotunde and R. Ibrahim, “A Review of Threat Modelling and Its Hybrid Approaches to Software Security Testing,” *ARPN Journal of Engineering and Applied Sciences*, 2015, pp. 17657–17664, 2015.

[39] D. Ghindici, G. Grimaud, I. Simplot-Ryl, Y. Liu, and I. Traore, “Integrated Security Verification and Validation: Case Study,” *IEEE Conference on Local Computer Networks*, 2006, pp. 1000–1007.

[40] A. D. Brucker and U. Sodan, “Deploying static application security testing on a large scale,” *GI Sicherheit*, 2014, pp. 91–101.

[41] L. Harjumaa and I. Tervonen, “Introducing Mitigation Use Cases to Enhance the Scope of Test Cases,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6434 LNCS, 2010, pp. 337–353.

[42] I. A. Tondel, M. G. Jaatun, and P. H. Meland, “Security Requirements for the Rest of Us: A Survey,” *IEEE Software*, vol. 25, no. 1, pp. 20–27, Jan. 2008.

[43] P. Stephanow and K. Khajehmoogahi, “Towards Continuous Security Certification of Software-as-a-Service Applications Using Web Application Testing Techniques,” *31st International Conference on Advanced Information Networking and Applications)*, 2017, pp. 931–938.

[44] E. J. Weyuker and F. I. Vokolos, “Experience with performance testing of software systems: issues, an approach, and case study,” *IEEE Transactions on Software Engineering*, 2000, vol. 26, pp. 1147–1156.

[45] G. Sindre and A. Opdahl, “Capturing security requirements through misuse cases,” *NIK 2001, Norsk Informatikkonferanse 2001*, p. 12, 2001.

[46] Z. Hui and S. Huang, “Comparison of SETAM with security use case and security misuse case: A software security testing study,” *Wuhan University Journal of Natural Sciences*, 2012, vol. 17, pp. 516–520.

[47] J. Jürjens, “Using UMLsec and goal trees for secure systems development,” *ACM symposium on Applied computing - SAC, 2002,* p. 1026.

[48] J. Bozic and F. Wotawa, “Security Testing Based on Attack Patterns,” *IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, 2014, pp. 4–11.

[49] C. B. Haley, R. Laney, J. D. Moffett, and B. Nuseibeh, “Security Requirements Engineering: A Framework for Representation and Analysis,” *IEEE Transactions on Software Engineering*, 2008, vol. 34, no. 1, pp. 133–153.

[50] L. Bulej, T. Bureš, V. Horký, J. Kotrč, L. Marek, T. Trojánek, and P. Tůma, “Unit testing performance with Stochastic Performance Logic,” *Automated Software Engineering*, 2017, vol. 24, pp. 139–187.

[51] L. Røstad, I. A. Tøndel, P. H. Meland, and G. R. Øie, “Learning by Failing (and Fixing),” *IEEE Security & Privacy Magazine*, 2008, vol. 6, pp. 54–56.

[52] A. B. Bondi and J. P. Ros, “Experience with Training a Remotely Located Performance Test Team in a Quasi-agile Global Environment,” *Fourth IEEE International Conference on Global Software Engineering*, 2009, pp. 254–261.

[53] E. W. Dijkstra, “Notes on structured programming,” *Structured programming*, 1972, pp. 1–82.

[54] D. Woodraska, M. Sanford, and D. Xu, “Security mutation testing of the FileZilla FTP server,” *ACM Symposium on Applied Computing*, 2011, p. 1425.

[55] L. Thomas, W. Xu, and D. Xu, “Mutation Analysis of Magento for Evaluating Threat Model-Based Security Testing,” *IEEE Annual Computer Software and Applications Conference Workshops*, 2011, pp. 184–189.

[56] M. Jovic, A. Adamoli, D. Zaparanuks, and M. Hauswirth, “Automating performance testing of interactive Java applications,” *5th Workshop on Automation of Software Test*, 2010, pp. 8–15.

[57] L. Williams, A. Meneely, and G. Shipley, “Protection Poker: The New Software Security ‘Game’;,” *IEEE Security & Privacy Magazine*, 2010, vol. 8, pp. 14–20.

[58] M. J. Johnson, E. M. Maximilien, C. W. Ho, and L. Williams, “Incorporating performance testing in test-driven development,” *IEEE Software*, 2007, vol. 24, pp. 67–73.

[59] T.-H. Chen, M. D. Syer, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora, “Analytics-Driven Load Testing: An Industrial Experience Report on Load Testing of Large-Scale Systems,” *39th International Conference on Software Engineering: Software Engineering in Practice Track*, 2017, pp. 243–252.

[60] V. Horký, P. Libič, L. Marek, A. Steinhauser, and P. Tůma, “Utilizing Performance Unit Tests To Increase Performance Awareness,” *6th International Conference on Performance Engineering*, 2015, pp. 289–300.

[61] B. Ferrell and R. Oostdyk, “Modeling and performance considerations for automated fault isolation in complex systems,” *IEEE Aerospace Conference*, 2010, vol. 9, pp. 1–8.

[62] L. ben Othmane, M. G. Jaatun, and E. Weippl, "*Empirical Research for Software Security Foundations and Experience*". Boca Raton: CRC Press, 2018.

[63] Y. Lincoln and E. Guba, “Naturalistic Inquiry,” *Encyclopedia of Research Design*, SAGE Publications, 2010.

[64] J. A. Maxwell, "Qualitative Research Design: An Interactive Approach", 3rd Ed., vol. 41. Sage Publications, 2012.

[65] M. Howard and S. Lipner, "The Security Development Lifecycle: SDL: A process for Developing Demonstrably More Secure Software". Microsoft Press, 2006.

[66] OWASP, “4.0 Testing Guide,” *OWASP foundation*, no. Cc, p. 224, 2014.

[67] B. Cartaxo, G. Pinto, and S. Soares, “The Role of Rapid Reviews in Supporting Decision-Making in Software Engineering Practice,” *22nd International Conference on Evaluation and Assessment in Software Engineering*, 2018, pp. 24–34.