UNDERSTANDING FACTORS AND PRACTICES OF SOFTWARE SECURITY AND PERFORMANCE VERIFICATION

Victor V. Ribeiro[1], Daniela Soares Cruzes[2], Guilherme Horta Travassos[1]

[1]Programa de Engenharia de Sistemas e Computação – Universidade Federal do Rio de Janeiro (UFRJ)
Caixa Postal 68.511 – Rio de Janeiro – RJ – Brazil

[2]SINTEF DIGITAL – SINTEF
Caixa Postal NO-7465 – Trondheim – Norway

vidigal@cos.ufrj.br, danielac@sintef.no, ght@cos.ufrj.br

Resumo. *Este artigo apresenta um corpo de conhecimento construído com base em evidência que caracteriza os requisitos não-funcionais mais relevantes para sistemas de software e as abordagens de teste de software que podem ser utilizadas para avaliar esses requisitos. O trabalho se especializa na caracterização de práticas de verificação de segurança e desempenho utilizadas em organizações de desenvolvimento de software e nos fatores que apoiam a tomada de decisão relacionadas à essas práticas. Adicionalmente, fatores de moderação que influenciam as atividades de verificação de segurança e desempenho e ações que apoiam a promoção desses fatores são apresentados. Os resultados apresentados são fortemente baseados em evidência, pois têm origem em diferentes estratégias de estudo e observações in vivo da indústria de software.*

Abstract. *This work offers an evidence-based body of knowledge characterizing the most relevant non-functional requirements for software systems, including suitable testing approaches to assess these requirements. The work goes more in-depth into characterizing security and performance verification practices in use in software development organizations and the factors that support decision-making regarding their use. Additionally, moderating factors of security and performance verification activities are presented, as well as actions to their promotion. The results are strongly evidence-based as they rely on different study strategies and in vivo observations at the software industry to support the findings. Software practitioners and researchers can benefit from using the body of knowledge for supporting their software projects and empirical investigations on non-functional requirements.*

1. Introduction

The importance of software systems to contemporary society increases specific concerns regarding some critical quality properties. Software engineers usually classify such properties as non-functional requirements (NFRs). NFRs represent software properties that are not related to the problem domain, such as security, performance,

usability, maintainability, portability. NFRs have always been essential to the success of software systems [Hammani 2014] [Ameller et al. 2012], but contemporary software systems have NFRs as essential properties [Joorabchi et al. 2013] [Rashid et al. 2015].

Despite several technologies supporting software development, this is a human-dependent activity and, therefore, error-prone. Therefore, as software systems should meet NFRs, the software development organizations include quality assurance activities throughout the software life cycle to evaluate these properties, preventing the occurrence of failures after software release. Therefore, the overall motivation of this work is summarized as follows: (1) the importance of non-functional requirements for software systems; (2) the need to include quality assurance activities (verification) aiming to assess if the software meets NFRs.

This work is divided into two cycles of research. The first one (section 4) focuses on identifying and understanding the most relevant NFRs for software systems and the testing techniques that can be applied to those NFRs. A body of knowledge consolidates the results of this first research cycle[1] [Ribeiro and Travassos 2016].

The second investigation cycle (sections 5 and 6) focuses on security and performance (S&P) verification. We choose these two specific NFRs because they were identified as the most relevant and because there was a request from Norwegian software companies to investigate them. Besides, the software development industry has strongly influenced this research cycle as we investigated the issues surrounding S&P verification through a case study with four different organizations in Brazil. The results are available through an evidence briefing[2][3], allowing a better understanding by practitioners, and partially published on Ribeiro, Cruzes, and Travassos [2018].

2. Research Goals and methodology overview

The research goal in its broader scope is to characterize the state of the practice regarding NFRs verification. Specific goals are following listed:

- Propose a Body of knowledge characterizing relevant NFRs and the software techniques that can be used to assess such requirements (NFR-BoK);
- Identify and characterize the S&P verification practices used by software development organizations;
- Identify the decision-making factors related to S&P verification used by software development organizations;
- Identify the moderator factors influencing the S&P verification;
- Identify actions used to promote S&P moderator factors.

Table 1 presents an overview of the methodology used to archive these goals.

---

[1] http://lens-ese.cos.ufrj.br/NFRWIKI

[2] http://lens-ese.cos.ufrj.br/spvsurvey/moderators-presentation.pdf

[3] http://lens-ese.cos.ufrj.br/spvsurvey/moderators-presentation-ptbr.pdf

Table 1. Methodology overview

Research methodology	Goals
Structured literature reviews	▪ Identify the most relevant NFRs ▪ Identify testing approaches to evaluate NFRs ▪ Build a body of knowledge of NFRs and testing approaches
Case study	▪ Identify and characterize S&P verification practices ▪ Identify S&P decision-making factors ▪ Identify and understand moderator factors influencing S&P verification
Rapid Reviews	▪ Improve the confidence of moderator factors
Survey	▪ Endorse our understanding of case study with case study participants ▪ Confirm the moderator factors pertinence with practitioners

3. Research approach, primary results, and contributions

Two investigation cycles with six steps compose this research (Figure 1). The scope of the first investigation cycle was related to software testing approaches supporting the assessment of NFRs. Thus, we use the technical literature to gain a better understanding of non-functional testing approaches.

However, as we gained knowledge on the topic, we realized that it would not be feasible to investigate all NFRs in-depth and that software testing is not a suitable approach to assess some NFRs. Therefore, in the second investigation cycle, the scope of this work has been adjusted to focus on security and performance and to encompass static verification activities (including software reviews).

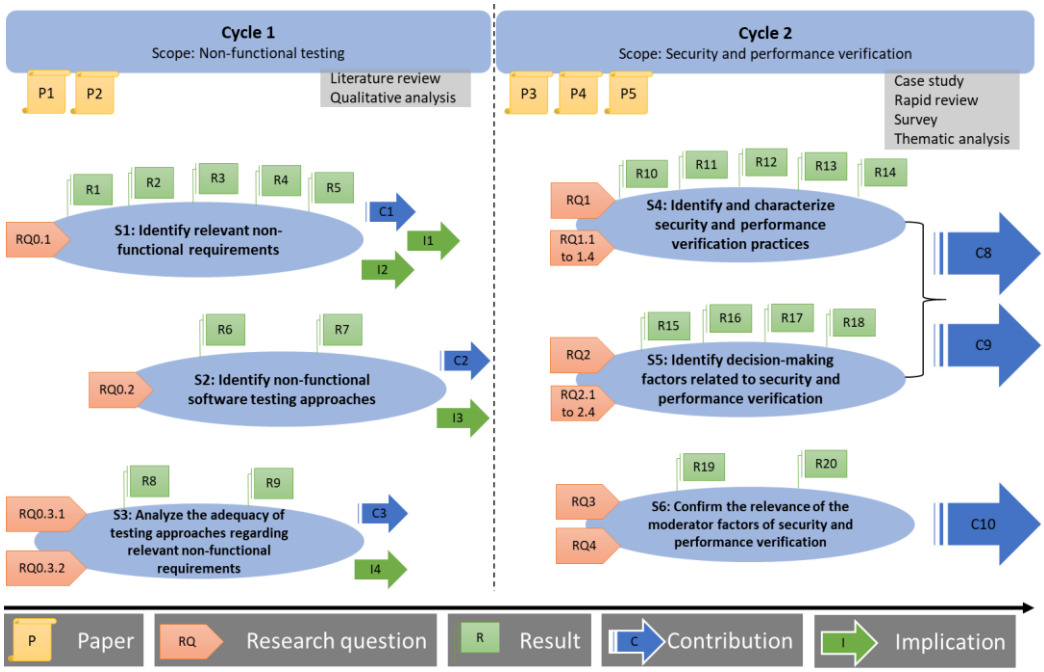

Figure 1. Research steps overview

4. Testing non-functional requirements: A body of knowledge

A body of knowledge (NFR-BoK) consolidates the results of the first investigation cycle. The NFR-BoK organizes information about identified NFRs, including the testing approaches that can be used to assess each of them. It is organized as a wiki to facilitate user navigation. Figure 2 presents the relevant NFRs so that by clicking on that, the user can view a page of detailed information. The first numerical character inside the brackets represents the number of papers that identify the NFR as relevant, and the second represents the number of testing approaches to assess it. For instance, six papers cite confidentiality as a relevant NFR, and there are two testing approaches to assess it.

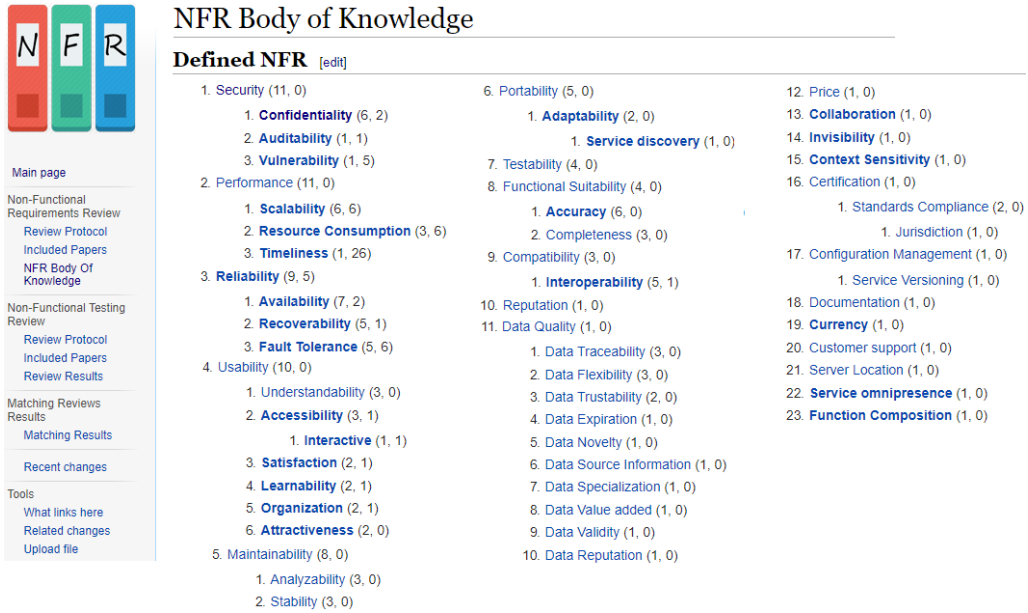

Figure 2. NFR-BoK - Relevant non-functional requirements

The detailed information about each NFR includes the following attributes:

- **Definition:** an NFR description explaining some system's capability, e.g., performance: It is the system capacity to provide appropriate use of resources (memory, CPU) needed to perform full functionality under stated conditions.
- **Synonyms:** names that present the same meaning, e.g., reliability is presented as a synonym of dependability.
- **Composed by:** other NFRs that are part of the main NFR, e.g., scalability, resource consumption, and timeliness compose the performance requirement.
- **Target object:** system element through which the NFR can be observed. Examples of target objects of the performance NFR: (1) system performance (how the system is using memory during execution), (2) function performance (what is the response time of specific function observing the messages among system functions), (3) interaction with user performance (response time observing user request and time until response).

- **Observed through:** how the NFR can be observed or how the software exposes it. For instance, performance can be observed through resources' monitoring or time observation in execution time.
- **Specification examples:** suggest how to specify an NFR, e.g., usability can be observed through user feedback.
- **Operationalization:** describes the mechanisms used to operationalize the NFR. An example of security operationalization is to store the password encrypted.
- **Risks:** risks related to non-compliance with an NFR. E.g., risks of availability requirement: loss of business opportunities or slow productivity.
- **It contains behavior NFR:** defines if an NFR represents a software behavior, e.g., "system services must response every request at most one second." Behaviors properties can be observed in execution time; they can be tested.
- **It contains representational NFR:** represents syntactical or technical software properties, e.g., "Software must use MySQL database." Representational properties are static properties, and so they cannot be tested. However, it can be assessed through static techniques such as inspections.
- **Assessable through testing:** defines if the NFR is testable. It is yes if the NFR represents a system behavior.
- **Who is affected by:** the roles directly affected by the NFR. E.g., Internal Stakeholders, Owner, Manager, Software Engineer, Programmer, Final User.
- **Mentioned by:** list of papers identifying the NFR, but not describing it.
- **Defined by:** list of papers identifying and describing the NFR.

The detailed information about each NFR testing approaches includes the following attributes:

- **Reference information** and **Abstract:** directly extracted from the paper that proposes the testing approach. It is used to find the paper that proposed the testing approach;
- **Proposal:** a brief description of the testing approach. It is useful to contextualize NFR-BoK's user.
- **System Domain/Type**: represents the context the testing approach was proposed
- **Software test step, Test level, and Test technique:** describe the testing dimensions covered by the testing approach;
- **Evaluation:** the kind of study was used to evaluate the testing approach;
- **Non-functional requirements covered:** presents the NFRs the approach is able to assess.

5. A perception of the state of the practice of security and performance verification

This section presents the S&P verification practices and their characterization regarding techniques, the definition of done criteria, automation level, and assets. Figure 3 presents a brief overview of the practices supporting S&P verification.

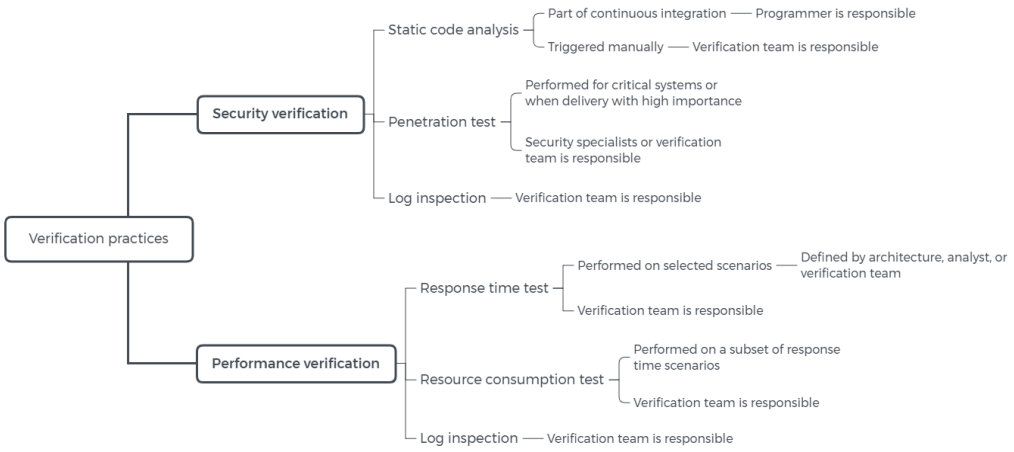

Figure 3. Identified security and performance verification practices

5.1. Details of software security and performance verification practices

Figure 4 presents the characterization of identified security verification practices regarding their *techniques*, *the definition of done*, *automation level*, *and assets*. Figure 5 presents the same information regarding performance practices.

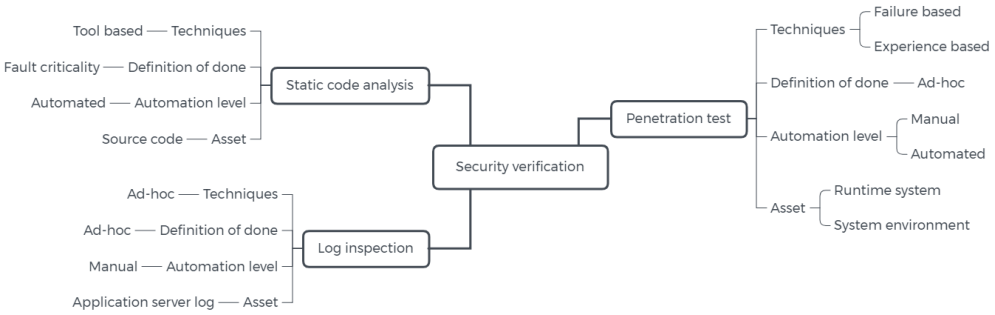

Figure 4. Software security verification practices details

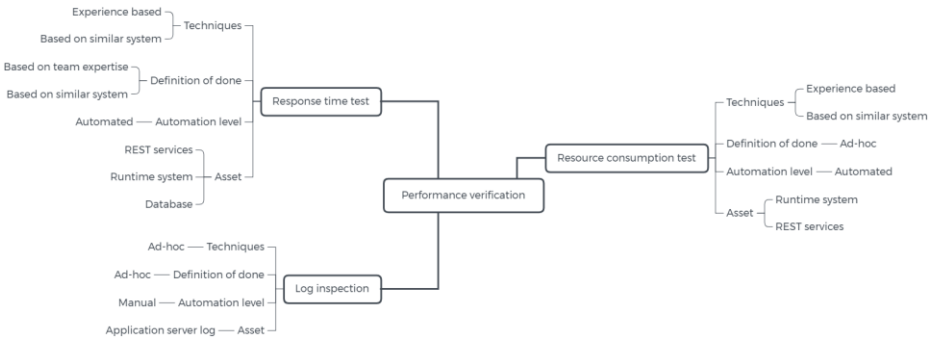

Figure 5. Software performance verification practices details

6. Moderator factors of security and performance verification

This section presents the eight moderator factors influencing security and performance verification. Besides, it presents a set of actions to promote each of the factors. Such factors emerged from the practice (case study), and then their pertinence (columns # and %) was confirmed through the opinion of practitioners (survey).

Table 2. MF1: Organizational awareness of S&P importance

MF1: Organizational awareness of security and performance importance		
Need for support from every stakeholder		
Need for training		
Actions to promote *organizational awareness of security and performance importance*	**#**	**%**
Keeping programmers well-informed about security and performance	28	90%
Promoting training	25	81%
Informing the customer about the real state of software security and performance	19	61%
New actions to promote *organizational awareness of security and performance importance*		
Simulation of security and performance failures and show business impact		
Regular meetings to discuss security practices		
External audit to mitigate human problems		
Having an ethical hacker would be extremely good for security and creating performance indicators		

Table 3. MF2: Cross-functional team

MF2: Cross-functional team		
Dependence on specialized verification team		
Dependence on database team		
Support of infrastructure team		
Support of legislation experts		
Actions to promote the build of a *cross-functional team*	**#**	**%**
Building a team having multiple skills	23	79%
Disseminating the view that the verification team is not the enemy but allied	23	79%
Stimulating interaction between members of different teams	18	62%
New actions to promote the build of a *cross-functional team*		
The team should have leaders swapping places (for example, marketing and development). team leaders can get to know limitations, capabilities, and point of view which can lead to better teamwork and results		
Highlight the positive results of having a multidisciplinary team		
Knowing what is problematic in other sectors of the organization		
Encouraging integration between teams working on similar topics		
Value verification professionals		
Select qualified people for the position		
Invest in the training and qualification of the verification team		
Apply Scrum		

Table 4. MF3: Suitable requirements moderator factor

MF3: Suitable requirements
Lack of well-defined requirements

Verification team should participate in the requirements phase		
Actions to promote the building of *suitable requirements*	**#**	**%**
Using techniques to handle security and performance requirements	25	81%
Involving the verification team in the requirements phase	24	77%
Stimulating the verification team to assess the testability of requirements	24	77%
New actions to promote the building of *suitable requirements*		
Involving the verification team in all phases of the software life cycle		
The verification team and Product Owner should discuss the specification to identify and adjust any deviations before the specification goes into development.		
Infrastructure team should assess security and performance		
Involving the requirements team in verification activities		

Table 5. MF4: Suitable support tools moderator factor

MF4: Suitable support tools		
Support tools decrease the effort of manual activities		
Preference for using free tools		
Allow the verification team to suggest new tools		
Automated tools generate unsuitable reports		
Verification report should include essential information		
Actions to promote the selection of *suitable support tools*	**#**	**%**
Allowing the technical team to suggest and adopt support tools	24	77%
Using tools consistent with the verification team knowledge	22	71%
Supporting the use of free tools	13	42%
New actions to promote the selection of *suitable support tools*		
Providing training to the verification team to enable them to operate the adopted tools (5 participants)		
Institutionalize the use of tools		
Using industry best-practice toolsets		
Support from the tool provider		

Table 6. MF5: Suitable verification environment moderator

MF5: Suitable verification environment		
Verification performed on the unsuitable environment		
Verification team should be able to control the environment		
Virtualization technologies assist in instantiating verification environment		
Actions to promote the configuration of the *suitable verification environment*	**#**	**%**
Using virtualization technologies to simulate execution environment	26	84%
Keeping the verification team well-informed about used technologies	22	71%
Using virtualization technologies to set up tests agents	19	61%
Performing each test case more than once and at a different period to mitigate external influences	16	52%

Scheduling the verification activities if it is not possible to instantiate a specific verification environment so that verification should never be performed in parallel with any other activity	15	48%
New actions to promote the configuration of the *suitable verification environment*		
Using automated verification		
Simulating a defined behavior that constitutes real user behavior		
Using techniques to generate suitable testing data		

Table 7. MF6: Suitable methodology moderator factor

MF6: Systematic verification methodology		
Lack of systematic verification techniques		
Organization methodology should be based on previously established standards		
Good methodology requirements		
Actions to promote the *systematic verification methodology*	**#**	**%**
Using a proposed methodology and adapting it to the context of the organization	21	78%
New actions to promote the *systematic verification methodology*		
Modify the company culture at some level by fostering a new methodology		
Search for a methodology aligned with stakeholders needs		
Use appropriately trained testers; avoid using a dopey methodology		
Create processes and revise them according to proposed methodology and company context		

Table 8. MF7: Security and performance planning moderator factor

MF7: *[Lack of]* Security and performance verification planning		
Security and performance verification requires extra effort		
Insufficient time to perform intended activities		
Actions to promote the *planning of security and performance verification*	**#**	**%**
Using a tool to guide the security and performance verification planning	25	86%
New actions to promote the *planning of security and performance verification*		
Including the security and performance verification activities as part of the development and maintenance cycle		
Having business knowledge helps prioritize the parts of the system that should be evaluated		

Table 9. MF8: Encouraging reuse practices moderator factor

MF8: Encourage reuse practices
Reuse of functional test cases
Reuse of previous systems test cases
Use of similar systems to determine requirements
Knowing common defects

Actions to promote the *reuse of S&P verification practices*	#	%
Knowing common defects (*e.g.*, vulnerabilities) and using pre-defined test cases to identify the failures caused by these defects	25	86%
Reusing the knowledge acquired from other similar systems as a basis for the definition of the requirements	23	79%
Reusing functional test cases as they represent real usage scenarios	19	66%
Reusing test cases from similar systems adapting parameters	13	45%
New actions to promote the *reuse of S&P verification practices*		
Creating a base of knowledge of recurring defects		
Mapping vulnerability according to the domain to promote the identification of vulnerabilities applicable to specific situations		
Functional test cases specify what could be added for performance verification		
Design real-time scenario with production volume data, per hour, per day transaction, Per week, among others.		
Reusing multiple test scenarios is very useful for both professional and runtime scenarios that we can insert in the context of similar new projects		

References

Ameller, D., Ayala, C., Cabot, J. and Franch, X. (Sep 2012). How do software architects consider non-functional requirements: An exploratory study. In *2012 20th IEEE International Requirements Engineering Conference (RE)*. IEEE. http://ieeexplore.ieee.org/document/6345838/.

Hammani, F. Z. (May 2014). Survey of Non-Functional Requirements modeling and verification of Software Product Lines. In *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*. IEEE. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6861085.

IEEE-610.12 (1990). 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology. *IEEE Computer Society*, v. 121990.

Joorabchi, M. E., Mesbah, A. and Kruchten, P. (Oct 2013). Real Challenges in Mobile App Development. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6681334.

Rashid, M., Ardito, L., and Torchiano, M. (Oct 2015). Energy Consumption Analysis of Algorithms Implementations. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7321198.

Ribeiro, V. V., Cruzes, D. S., and Travassos, G. H. (Nov 2018). A Perception of the Practice of Software Security and Performance Verification. In the *2018 25th Australasian Software Engineering Conference (ASWEC)*. IEEE.

Ribeiro, V. V., and Travassos, G. H. (2016). Testing Non-Functional Requirements: Lacking Technologies or Researching Opportunities. XV Brazilian Symposium on Software Quality.