Noname manuscript No.
(will be inserted by the editor)

Exploring Performance Assurance Practices and Challenges in Agile Software Development: An Ethnographic Study

Luca Traini

Received: date / Accepted: date

Abstract *Background:* Agile principles play a pivotal role in modern software development. Unfortunately, the assessment of non-functional software properties, such as performance, can be challenging in Agile Software Development (ASD). Agile mentality tends to favor functional development over non-functional quality assurance. Additionally, frequent code changes and software releases make impractical the use of classical performance assurance approaches.
Objective: This paper investigates the current practices, problems and challenges of performance assurance in a real context of ASD. To the best of our knowledge, this is the first empirical study that specifically investigate performance assurance in ASD daily work.
Method: Through a 6-months industry collaboration with a large software organization that adopts ASD, we investigated practical and management problems in handling performance assurance activities. The research was conducted in line with ethnographic research, which guided towards building knowledge from participatory observations, unstructured interviews and reviews of documentations.
Results: The study shows that the case organization still relies on a waterfall-like approach for performance assurance. Such an approach showed to be inadequate for ASD, thereby leading to a sub-optimal management of performance assessment activities. We distilled three key challenges when trying to improve the performance assurance process: (i) managing performance assessment activities, (ii) continuous performance assessment and (iii) defining the performance assessment effort.
Conclusions: The assessment of software performance in the context of ASD is still far from being flawless. The lack of guidelines and well-established practices induces the adoption of approaches that can be obsolete and inadequate

L. Traini
University of L'Aquila, Italy
E-mail: luca.traini@univaq.it

for ASD. Further research is needed to improve the performance management in this context, and to enable effective continuous performance assessment.

Keywords Software Performance Engineering · Ethnographic studies · Agile Software Development

1 Introduction

In the last decades, the principles behind the Agile Manifesto (Beck et al., 2001) have profoundly changed the way software is produced. Software development methodologies inspired by these principles (*e.g.,* Scrum and Kanban) are today widely adopted[1], and are gradually replacing the traditional waterfall approach. Indeed, Agile Software Development (ASD) better meets the dynamic nature of today's software development business and the significant pressure to deliver fast to the market (Rubin, 2012).

However, although ASD and DevOps have successfully enabled companies to address the current fast-to-market trend, their cost in terms of software quality is still disputable (Rubin, 2012). In the past years, researchers criticized ASD for mainly focusing on functional aspects and neglecting non-functional quality attributes (Ramesh et al., 2010; Inayat et al., 2015), and highlighted a lack of well-defined practices to effectively manage non-functional quality assurance (Alsaqaf et al., 2017, 2019; Behutiye et al., 2020a,b; Kasauli et al., 2021).

Short iteration cycles and time constraints minimize the focus on addressing non-functional requirements (Behutiye et al., 2020a; Alsaqaf et al., 2019), and hamper the adoption of traditional quality assurance approaches. Software performance assurance, for example, can be especially challenging in these contexts (Ramesh et al., 2010; Woodside et al., 2007).

In the past years, studies has been conducted to investigate challenges in non-functional quality assurance in the ASD context (Alsaqaf et al., 2017, 2019; Behutiye et al., 2020a,b). However, while these studies investigate non-functional attributes in general, there is still little knowledge on practices and challenges to assess specific non-functional quality attributes, such as performance.

Poor software performance impacts user engagement and satisfaction (Brutlag, 2009), wastes computational resources, and degrade system response time and throughput. Since the early days of ASD, researchers expressed concerns about how to inject performance assurance activities in Agile iterations (Woodside et al., 2007). Indeed, due to their time-consuming nature, common performance assurance practices (*e.g.,* load testing (Jiang and Hassan, 2015)) are often unsuitable for ASD. Moreover, mentality and principles behind Agile often contradict common performance engineering conjectures. For example, Fowler and Beck, namely two signatories of the Agile Manifesto (Beck et al., 2001), suggest to consider software performance only after making the software

[1] 15th State of Agile Survey. `https://bit.ly/3azEj5r`

clear and maintainable (Auer and Beck, 1996; Fowler, 2002), thereby conceptually reflecting the famous quote from Knuth that "*Premature optimization is the root of all evil in programming*" (Knuth, 2007). On the other hand, classical performance engineering literature argues that the "*fix-it-late*" attitude is one of the causes of major performance failures (Smith and Williams, 2001). To best of our knowledge, the only attempt to tackle this problem was made by Chih-Wei Ho et al. (2006), which proposed an evolutionary model for performance requirements specifications and corresponding validation testing that can be integrated into ASD.

However, after two decades of ASD, there is still a lack of empirical knowledge on performance assurance in this context. Although several studies have broadly investigated non-functional quality assurance in ASD, there is still little specific knowledge on the practices and challenges of performance assurance in ASD. Moreover, prior work present results based on data collected through interviews and surveys, but it does not examine how agile teams tackle non-functional quality assurance in their daily work.

In order to contribute to fill this gap, in this paper we present an empirical study investigating performance assessment practice and challenges in ASD daily work. The research was conducted by means of a 6-months ethnographic study (Sharp et al., 2016) in a Research & Development division of a large company[2], which uses the Scrum framework (Rubin, 2012) to support their agile practices. During the first three months, we gained understanding on the performance assurance practices and problems through the observation of daily work and meeting sessions, individual interviews, participation in demo sessions, process workshops and review of documentation. In the last three months, we have actively participated to the improvement of the performance assurance process. Ethnography focus on daily activities helped to capture a holistic view of the performance assurance process, and to distill promising research directions related to three broad challenges: *managing performance assessment activities*, *continuous performance assessment* and *determining the performance assessment effort.*

The rest of the article is structured as follows. Section 2 introduces the study context, *i.e.,* the case organization, and Section 3 outlines the study design. Section 4 describes the observed practices for performance assurance, and the problems that arose from the adoption of such practices. In Section 5 we present proposals that were discussed to improve the performance assurance process. Section 6 outlines three key challenges for the improvement of performance assurance in ASD along with interesting research directions. Section 7 discusses the consistency of our findings with those of previous studies reported in the literature. Section 8 describes threats to validity. Section 9 presents related work, and Section 10 concludes this paper.

[2] Due to the sensitivity of the results presented here-in, the organization chose to stay incognito. Therefore, in this paper, we use fictitious names of the company and the product.

2 Study context

This study was carried out in a Research & Development (R&D) division of *ECorp* [2], based in Italy, where more than 120 people work on building *MESPlatform* [2], *i.e.,* a software platform for the manufacturing industry domain. *MESPlatform* provides foundation for building Manufacturing Execution Systems (Meyer, 2009)(MES), which allow industries to digitalize their manufacturing chains by providing a real-time software layer to track and document the transformation of raw materials to finished goods. The software development division used the Scrum framework to support their agile practices (Rubin, 2012). In the following subsections we first provide a general description of the Scrum framework, then we describe how Scrum is implemented in the case company (number and size of agile teams, sprint length, etc.).

2.1 The Scrum framework

Scrum is a widely popular agile framework[3], which consists of a set of roles, events, artifacts, and rules that bind them together (Rubin, 2012). The three key roles in Scrum are the product owner, the Scrum master and the developer. The product owner is the central role in requirements management in Scrum, as he is solely responsible for prioritizing and managing the requirements (Heikkila et al., 2013) The Scrum master is accountable for establishing Scrum in the organization, and helping professionals to understand the theory and practice behind Scrum. Developers are, instead, accountable for turning requirements into actual working software. In Scrum, software development happens within fixed length events called sprints. Each one these events begin with a planning meeting to laying out the work to be performed for the sprint. This plan is created through the collaborative effort of an entire agile team (usually composed by some developers, a product owner and a Scrum master). In order to inspect progress within the sprint, each agile team performs short daily stand-up meetings to monitor and adjust the planned work.

The key requirements management tool in Scrum is the product backlog (Heikkila et al., 2013). The product backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases. Whenever a backlog item is added to the product backlog, the development team together with the product owner estimates the size of the item. Based on the size and importance of the new item, the backlog item is then prioritized into the product backlog. The product backlog usually includes different types on artifacts . Examples of backlog items include user stories and epics. The former represents the smallest unit of work in the Scrum framework. It is an informal, general explanation of a software feature written from the perspective of the end user. The latter, instead, represents a specific type of user story, which is too big to fit in less than one sprint. Epics are useful as placeholders for large requirements, and are usually

[3] 15^{th} State of Agile Survey. `https://bit.ly/3azEj5r`

progressively refined into a set of smaller user stories at the appropriate time (Rubin, 2012).

2.2 Scrum at *ECorp*

The software development division was composed by 13 agile teams. Each team had between 8 and 10 developers, one of whom played the additional role of Scrum Master. Three additional teams were responsible of non-functional software quality aspects: a UX team, a security team and an enterprise testing team. The latter was in charge of performing highly complex tests to assess important non-functional quality attributes, such as reliability, robustness and performance. One head of development, three product owners, one product manager, three software architects, one release manager, one quality assurance specialist and two Agile coaches were other professionals involved in the software development process. Seven agile teams were distributed in other countries, i.e. India and Romania, while others agile teams and professionals were co-located in the same building in Italy. Development followed 3-week sprints, or iterations, and the teams used Scrum ceremonies, *e.g.,* sprint planning meetings, stand-up meetings, demo sessions and retrospectives. Software release happened every 6 sprints (roughly every 4 months).

3 Study design

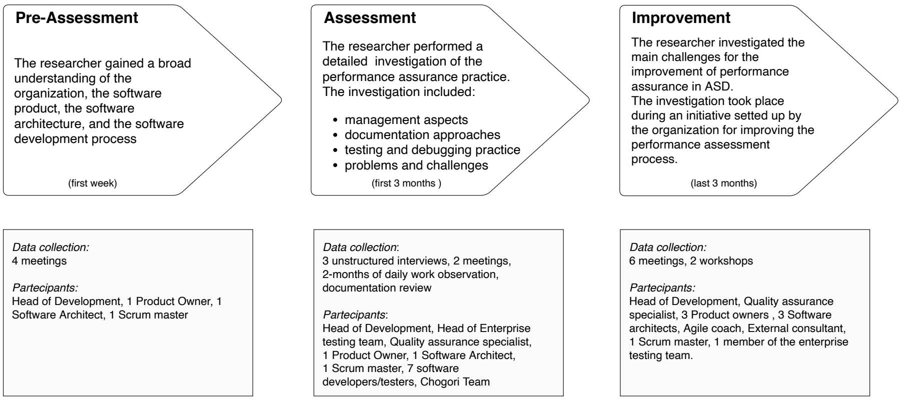
Previous studies on non-functional quality assurance in ASD are based on interviews and questionnaires, but they do not examine how agile teams tackle quality assurance in their daily work. This paper takes a different perspective: it explores performance assurances practices and challenges in ASD daily work by means of an ethnographic study (Sharp et al., 2016). The key strength of ethnography that is overlooked by other research methods is the support it provides to explicate the rationale beyond practice from an insider's point of view to capture both *what* practitioners do in a context, and *why.* Ethnographic studies often focus on the *ordinary detail of life* (Anderson, 1997). The importance of attending to the ordinary details of life lies in the role these details play in relation to how the everyday tasks are addressed (Sharp et al., 2016). The result of an ethnography is often more comprehensive and detailed when compared to results obtained with other research methods. This is because it aims to communicate the broad picture. This comprehensive and detailed set of data is often referred to as a "*thick description*" (Geertz, 1988) of the community and culture studied.

In order to formalize the study, we used the five ethnographic dimensions, as proposed by Sharp et al. (2016):

1. observation type (participant or not) was mixed. In the first phase of the study non-participant observation was used, with the researcher asking questions and observing individuals performing their tasks; In the second

phase of the study, the researcher actively participated to meetings with the goal of improving the process;

2. Duration of field study was 6 months;
3. Space and location where the observation happened was the R&D labs of *ECorp* where *MESPlatform* was developed (Italy);
4. No specific theoretical underpinnings were used;
5. The ethnographer intent was to understand the current process for performance assurance and the challenges in improving this process;



Pre-Assessment

The researcher gained a broad understanding of the organization, the software product, the software architecture, and the software development process

(first week)

Assessment

The researcher performed a detailed investigation of the performance assurance practice. The investigation included:

- management aspects
- documentation approaches
- testing and debugging practice
- problems and challenges

(first 3 months)

Improvement

The researcher investigated the main challenges for the improvement of performance assurance in ASD.
The investigation took place during an initiative setted up by the organization for improving the performance assessment process.

(last 3 months)

Data collection:
4 meetings

Partecipants:
Head of Development, 1 Product Owner, 1 Software Architect, 1 Scrum master

Data collection:
3 unstructured interviews, 2 meetings, 2-months of daily work observation, documentation review

Partecipants:
Head of Development, Head of Enterprise testing team, Quality assurance specialist, 1 Product Owner, 1 Software Architect, 1 Scrum master, 7 software developers/testers, Chogori Team

Data collection:
6 meetings, 2 workshops

Partecipants:
Head of Development, Quality assurance specialist, 3 Product owners , 3 Software architects, Agile coach, External consultant, 1 Scrum master, 1 member of the enterprise testing team.

Fig. 1 Overview of the study. The upper part of the figure briefly describes the three main phases of the study. The lower part reports for each phase the activities performed to collect data and professionals involved these activities. A more detailed description of the activities performed during the study can be found in Table 1 (Pre-Assessment), Table 2 (Assessment) and Table 3 (Improvement).

Fig. 1 provides an overview of the three main phases of the study: *Pre-Assessment*, *Assessment* and *Improvement*. The study started with four individual meetings to gain an overview of the software development process, the product and the software architecture (*Pre-Assessment*). These meetings involved the head of software development, a product owner, a scrum master and a software architect (see Table 1 for further details).

After these preliminary meetings, the investigation on performance assurance practices started (*Assessment*). The first unstructured interview was conducted with the head of development to gain an overview of the current process for performance assurance and the parts involved in this process (e.g., teams and professionals). After this interview, the researcher agreed with the head of development to spend two months of daily work observations within an agile team in charge of performance testing activities. The researcher observed the team in their daily work activities, such as scrum ceremonies (e.g, stand-up meeting, sprint plan and retrospectives) and technical activities (e.g., performance testing and debugging). Other two unstructured interview sessions were carried out with the quality assurance specialist to gain a detailed picture of the performance assurance process. Additionally, the researcher participated to all the meetings that were related to performance assurance aspects. These meetings involved agile teams, product owners, software architects, quality assurance specialists and the enterprise testing teams (see Table 2 for further details). The data was collected through different media like handwritten notes, photographs and digital copies of documents and artifacts.

After the first three months, the head of development proposed to start an investigation to identify opportunities for improvement in the performance assurance process (*Improvement*). To this aim, a series of meetings involving product owners, software architects, quality assurance specialist, head of development, a scrum master, an external consultant and the enterprise testing team were held (see Table 3 for further details). The researcher actively participated to these meetings, thereby influencing the outcome of the research.

After these meetings, two activities were identified to improve the performance assurance process. The researcher actively participated to both these activities, which mainly involved meetings with product owners and software architects.

In order to complement data gathered from interviews, meetings and observations, the researcher also analyzed process and software documentation in Confluence[4] and Azure DevOps Server[5](ADS), which supported the R&D division in holding all the software development information (e.g., product backlog, test plans and wikis).

The collected data were then analyzed to derive (i) the practices adopted by the organization for performance assessment along with their problems, and (ii) the challenges in improving the performance assessment process.

Partecipants	**Type**	**Description**
Head of Development	Meeting	Overview of the organization structure, software product (and domain) and software development process.
Product Owner	Meeting	Overview of to the software product.
Scrum Master	Meeting	Overview the Scrum framework.
Software Architect	Meeting	Overview to the software architecture.

Table 1 Data collection activities in the Pre-Assessment phase.

[4] Atlassian Confluence `https://www.atlassian.com/software/confluence`

[5] Microsoft Azure DevOps Server, `https://azure.microsoft.com/it-it/services/devops/server/`

Partecipants	**Type**	**Description**
Head of Development	Unstructured interview	Overview of the performance assurance process (*e.g.,* stakeholders and test frequency).
Chogori Team	Daily work observation	2-months of daily work observations. The observations involved all the activities performed by the Chogori team: including scrum ceremonies (*e.g.,* sprint planning, sprint retrospectives, daily standup meetings), technical activities (*e.g.,* performance testing and debugging) and meetings related to performance assessment.
Quality Assurance Specialist	Unstructured interview	Broad description of the performance assurance process: stakeholders involved, how information related to performance assurance flows within the organization.
Quality Assurance Specialist	Unstructured interview	Detailed description of the performance assurance process: documentation approaches, PR ownership and prioritazion mechanism.
Head of Enterprise Testing Team, 1 Product owner, Quality Assurance Specialist, 1 Scrum Master, 1 Software Architect	Meeting	Alignement on the NFR validation process.
7 Software developers/testers	Meeting	Periodic meeting of the community of practice for quality assurance.
-	Documentation Review	Review of both process and technical documentation, including 92 PRs, test plans (24 test cases and 6 user stories), wikis and others product backlog items.

Table 2 Data collection activities in the Assessment phase.

Partecipants	**Type**	**Description**
Head of Development, 1 Product owner, Quality Assurance specialist	Meeting	Broad discussion on the main problems affecting the performance assurance process, and promising ideas for improvement.
Head of Development, 3 Product owners, Quality Assurance specialist, 1 Scrum Master	Meeting	Discussion and analysis of the main problems of the performance assurance process.
Agile Coach	Meeting	Discussion of the main problems affecting the management of performance assessment activities.
Quality assurance specialist, External consultant	Meeting	Discussion on how to improve the performance assurance process.
Quality assurance specialist, 1 member of the enterprise testing team	Meeting	Discussion on the lack of alignment between PR priority and performance test execution.
3 Product owners, Head of Development, 1 Quality Assurance specialist, 1 Scrum Master	Meeting	Discussion and definition of the next steps for the improvement of performance assurance.
3 Product owners, 3 Software architects, 1 Scrum master	Workshop	Refinement of the PR list (87 PRs kept and 5 PRs discarded).
2 Product owners, 3 Software architects, 1 Scrum master	Workshop	Selection of an automated performance testing suite (12 PRs selected, involving 7 web test PRs and 5 load tests PRs).

Table 3 Data collection activities in the Improvement phase.

4 Performance assessment practices

The observed performance assessment activities still followed a waterfall-like approach. In this section, we first describe the performance assessment process, as it was observed in the case organization, and then we report problems that arose from the adoption of such process.

4.1 Description

In the following, we outline the main components of the performance assessment process. Specifically we describe (i) the Performance Requirements (PR) *MESPlatform* was subject to, (ii) the process used to manage performance assessment activities, (iii) the documentation approach and (iv) the PR verification process.

4.1.1 Performance Requirements

MESPlatform was subject to 92 PRs, where each PR involved a set of testing scenarios that were used to evaluate the performance of a specific system operation. Each testing scenario was usually associated to a target (e.g., minimal expected throughput, maximal expected response time), which was defined according to customer agreements, or based on the system knowledge of product owners and software architects. In some PRs no targets were specified. These PRs were called benchmarks and were used to assess performance of new functionalities or to compare the performance of particular system operations across different software versions. Each PR described the system operation under test and the configurations required to each testing scenario, such as virtual machine specifications (e.g., number of CPU cores, memory size), database size (i.e, records stored in the database) and number of concurrent users. Overall there were more than 300 testing scenarios across the 92 PRs. Some of these tests required just few seconds of execution while others required more than one day. PRs involved four different types of performance tests: UI tests, web tests, load tests and mixed tests. Performance UI tests were used to measure the time needed to execute a particular task in the *MESPlatform* UI by simulating real user interaction through Selenium [6]. The advantage of these tests was their representativeness of the true usage of the application, as the web application is actually executed in a browser, thereby considering also UI rendering time in the response time measurement. Unfortunately, these tests are usually demanding in terms of maintenance, as even a tiny UI change may corrupt them. According to a senior software tester, it was rare that UI tests were used without additional maintenance tasks from one release to another. Performance web tests, instead, involved a series of HTTP requests that simulated a single user interaction. In this type of test, only server side response time (or throughput) was measured, while the UI rendering time was ignored. Load tests (Jiang and Hassan, 2015) were used to simulate many users that interact with the system at the same time. Load tests involved the simultaneous execution of multiple web tests that simulate multiple users making multiple simultaneous HTTP requests. Mixed performance tests, instead, were UI tests combined with load tests, which enabled response time measurements of UI user interactions on the system under load.

[6] Selenium WebDriver, `https://www.selenium.dev`

4.1.2 PRs management

New PRs were usually defined by individual product owners or software architects based on their system knowledge or according to customer needs. On the other hand, the deprecation of a PR usually required acknowledgements among multiple architects and product owners. Overall, performance assurance activities still followed a waterfall-like process, i.e. performance tests were executed before each software release. At the best case, every PR was tested once before each release, however, due to time and resources constraints, the exhaustively assessment of PRs before release was usually impractical. As a matter of fact, at the time of study, no more than 35% of PRs were addressed per release (see Fig. 2).

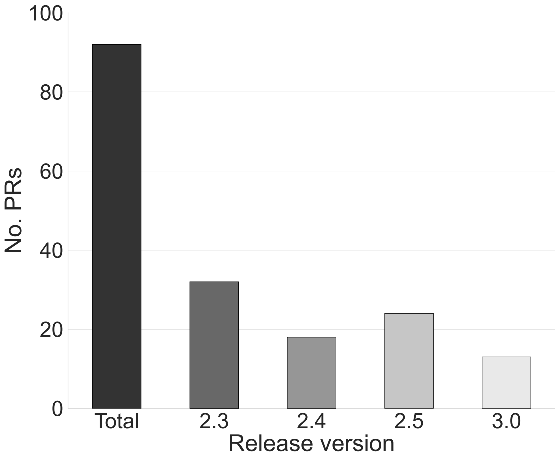


Fig. 2 PRs tested per releases. The first bar represents the total number of PRs, while the others represent PRs tested in a particular release version.

In order to ensure the assessment of relevant PRs before releasing, the company employed a priority mechanism, where each PR was prioritized with a number ranging from 1 to 4, with 1 being the highest priority and 4 the lowest. PR priorities were usually not stable across releases, since the development activities performed within a release cycle may often change the relevance of some PRs. According to the quality assurance specialist, priorities were usually updated before each release in a long one-day meeting involving software architects and product owners.

Performance tests were usually performed by the enterprise testing team, which chosen performance tests to execute according to priorities and time/resource constraints. In some cases, product owners specifically asked mandatory assessment of certain PRs. Some PRs were also assigned to a "special" agile team, called *ChogoRi*, which performed complementary performance assurance activities along with feature development and other operational tasks.

4.1.3 Documentation

PRs were documented through a custom artifact of ADS[7], which was specifically created to document Non-Functional Requirements (NFR). The catalog of PRs was accessible through ADS, and each PRs artifact contained an ID, a title, a description and the software versions in which the NFR was tested. The description contained the performance testing type (e.g., UI, load test), the description of system operation(s) under test, and a set of testing scenarios along with their detailed descriptions (e.g., machine specs, number of concurrent users). Supplementary information on PR was held in Confluence wikis. The enterprise testing team used test plans in ADS to manage and document the execution of performance tests, while the *ChogoRi* team adopted user stories[8]. Both test plans and user stories contained a reference to the PR ID.

4.1.4 PR verification

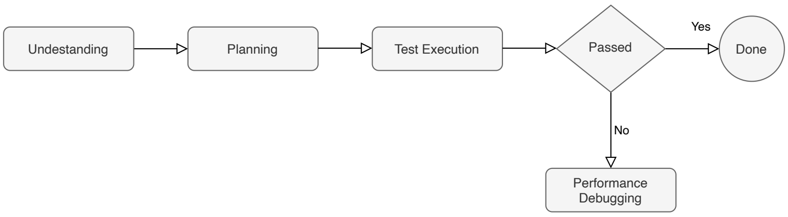


Fig. 3 PR verification process.

Fig. 3 outlines the main phases of PR verification, as they were observed in the *ChogoRi* team daily work. In the first phase, the team analyzed the PR description to gain understanding on the required testing scenarios. Often, in case of ambiguous PR description, architects or software developers helped the team in this process. In the planning phase, the work needed for PR verification was divided into tasks, such as environment configuration or test development, and assigned to team members. In the third phase, performance tests were executed while collecting application logs and performance indices

[7] Test plans, Azure DevOps Service. `https://bit.ly/3aRlmdv`

[8] User story, Agile Alliance. `https://bit.ly/369HxtY`

(e.g., throughput, response time, CPU and memory utilizations). The operational data was then analyzed by the team to identify relevant anomalies and to verify PRs targets. In the case of benchmark PR (*i.e.,* no target specified), performance test results were typically compared to a baseline, *e.g.,* results of the latest release tested. If results did not met targets or showed relevant performance deviation compared to baseline, they were reported to architects and/or product owners, which, after a careful evaluation, could confirm the performance bug. Similarly to functional bugs, performance bugs were tracked in ADS along with a level of severity[9]. The severity defines whether the performance bug had to be immediately resolved or it could be addressed in later development stages (according to priorities).

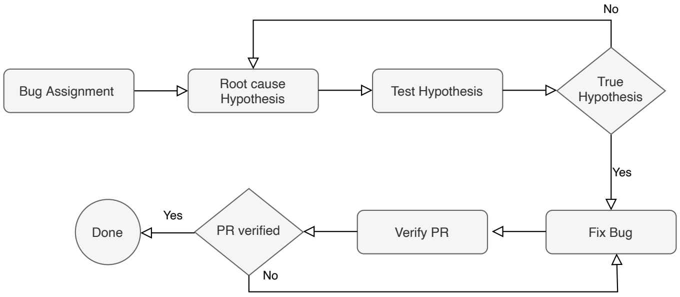


Fig. 4 Performance debugging process.

Fig. 4 outlines the performance debugging process as observed in the *ChogoRi* team. In order to assign the bug to the proper development team, the system components that are affected by the bug had to be identified. To this end, the *ChogoRi* team analyzed performance indices and/or log statements, that are somehow related to the performance issue, to pinpoint the software components that are affected by the performance bug. Often, the knowledge on the system by the *ChogoRi* team might not be sufficient to perform this task. Hence, in these cases, they were helped by software architects and developers. Once the affected components were identified, the bug was assigned to the proper development team, which thoroughly study the bug symptoms to identify the likely root cause. The root causing process usually involved three steps: 1) making a root cause hypothesis, 2) changing the source code according to the hypothesis and 3) validating the hypothesis by re-running performance tests on the modified system snapshot. Often this process had to be repeated several times in order to identify the real root cause. Once the root cause has been determined, the development team actually resolves the bug, and subsequently the performance testing team re-reruns testing scenarios to verify the PR. If the PR was met, then the bug could be marked as “resolved”.

[9] Azure DevOps Service, bug management. `https://bit.ly/3thSZgZ`

4.2 Problems

In the following, we report the main problems that were observed within the performance assurance process. We categorize these problems in two broad categories: PR management and lack of early feedback.

4.2.1 PR management

After a thorough analysis of test plans, user stories and PRs, we noticed that PR management was not working as expected. Although the number of PRs continued to grow from one release to another, due to the new features added to *MESPlatform*, most of them were often not verified. The cause of this phenomenon was that, while adding a new PR was a relatively cheap process for the organization (since it required the decision of single product owner or architect), the deprecation of a PR required the agreement of multiple product owners and architects. As a consequence, product owners kept adding PRs but their deprecation rarely happened, and, from time to time, the PRs list became unmanageable. Moreover, the process overhead introduced by this high number of PRs was often not justified by their usefulness. For example, we found that 35 PRs were never tested in the last 6 releases (see Fig. 5), i.e., since ∼2 years.

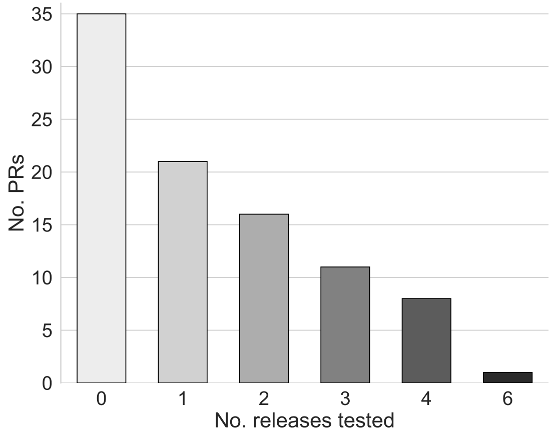


Fig. 5 Number of times PRs have been tested in the last 6 releases. The x label represents the amount of releases tested, while the y label represents the number of PRs that have been tested x times in the last 6 releases.

The increased number of PRs also impacted the prioritization mechanism which was showing its vulnerability. Indeed, the analysis of the test plans and user stories reported a lack of alignment between priorities and what was actually tested (see Fig. 6). For example, several PRs with priority 1 (i.e., the highest) were not tested in the last release, while others with priority 4 (i.e., the lowest) were. Specifically, the PRs tested involved 7 out of 11 PRs with priority 1, 2 out of 18 PRs with priority 2, 1 out of 19 PRs with priority 3, and 3 out of 44 PRs with priority 4. When the researcher asked the quality assurance specialist why this happened, she replied that meetings for updating priorities were not held in the last two releases, therefore priorities might be obsolete. Indeed, product owners and software architects were usually overloaded with many relevant tasks in the organization, hence keeping them busy for a whole work day to update NFRs priorities was unfeasible in the last releases.

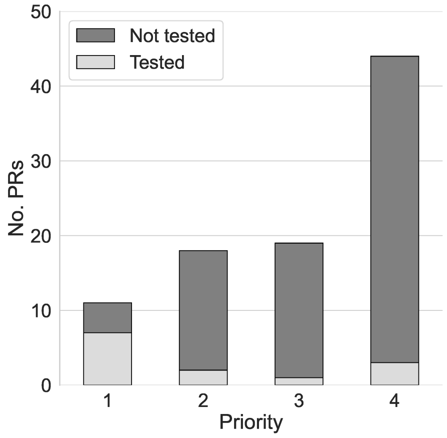


Fig. 6 Number of PRs tested (and not tested) in the latest release, grouped by priority.

Nevertheless, according to the quality assurance specialist, at that time, the criteria used to select PRs for a particular release was not clear. We asked clarification to the enterprise team and product owners, and it turned out that several PRs were autonomously chosen by the enterprise testing team, apparently, according to their practical convenience. The lack of well established practices to manage PRs led to a suboptimal selection of performance tests, which compromised the performance assessment of *MESPlatform*. As a matter of fact, at the time of the study, several informants in the organization reported that other R&D divisions in *ECorp* that used *MESPlatform* to build other products, were experiencing severe performance issues. Moreover, while analyzing the product backlog, we found that an entire epic was devoted to improving *MESPlatform* performance, hence suggesting that a massive rework was planned. Note that epic is the larger unit of work in Scrum, and it is used as a placeholder that represents work that cannot be finished within a sprint (*i.e.,* 3 weeks).

4.2.2 Lack of early feedback

While functional tests were continuously executed, both in the continuous integration pipeline and in nightly builds, performance tests were executed at most once per release. Nevertheless, from one release to another, thousands of code changes were performed that might potentially affect software performance. This lack of performance feedback during development might hide potentially harmful performance issues, which could eventually lead to expensive maintenance activities. Moreover, a higher number of code changes usually implies a higher number of potential causes of performance regression, which makes harder analysis and problem resolution. In order to provide a better understanding of this problem, in the following we report the challenges observed by the researcher, in the *ChogoRi* team, during a complex PR verification.

The PR was derived from a real world scenario of an *MESPlatform* customer, and it involved a set of load testing scenarios. For each scenario, a target was specified to define the minimal expected throughput for a specific system operation. After setting up the environment (i.e., the virtual machine, the *MESPlatform* instance and its databases) and launching performance tests, the tester found that PR targets were not met. On the other hand, in the previous release, the same PR was verified without reporting issues, which implied that performance regression was introduced during the development of last release. The analysis of performance indices and logs found that the response time of the analyzed operation kept increasing during the test, and, in the meantime, the CPU utilization temporarily drop to zero. The symptoms of the performance issue were then reported to software architects and product owners, which, after a careful analysis, classified the problem as a critical bug that should have been immediately resolved. The bug was assigned to a development team who had worked, during the last release development, on the components apparently affected by the performance issue. Unfortunately, these software components had been subject to several changes since the previous release, thereby implying a potentially large set of potential root causes. As a matter of fact, the truly identification of the root cause required several attempts and about a week of work, in which the development team performed experimental code changes to the system snapshot and the performance testing team re-launched tests against the system snapshot to evaluate its performance. An earlier performance feedback would have probably implied less iterations to isolate the problem, due to a lower number of potential code changes, thereby enabling a faster root cause detection.

The late detection of the bug also impacted the effort and the quality of debugging. The performance bug was caused by a well known problem in .NET [10] asynchronous programming called “threadpool starvation” [11]. The root cause was the introduction of a synchronous call in the critical path of a system operation. Unfortunately, after the introduction of the bug, several changes were

[10] Microsoft .NET. `https://bit.ly/2Muc3If`

[11] NET Core, ThreadPool Starvation. `http://bit.ly/3ozYnII`

performed on the system that made difficult to fix the synchronous call. The impact of the bug was partially hampered through a work-around[11], which allowed the PR tests to pass. However, due to time-pressure and rework complexity, the actual resolution of the bug was unfeasible. An earlier identification of the problem would have probably enabled software developers to be more aware of the potential impact of their design decision on software performance, thereby avoiding potential technical debt.

5 An initiative to improve performance assessment

During the last three months of this empirical study, a series of workshops and meetings were held with the goal of improving the performance assessment process. Several proposals were discussed to tackle the problems faced by the organization, some of them are briefly discussed in the following.

One proposal was to assign performance assessment activities directly to development teams to enable early assessment of development activities as soon as they are completed. The proposal was discarded due to various reasons. For instance, according to the head of development, development teams were often too busy to deal with performance testing activities. Moreover, according to several informants, they often did not have the required technical knowledge to reliably perform them. Furthermore, many PRs referred the whole system architecture, hence they usually depended from the simultaneous development activities of multiple teams.

Another problem discussed was the decreasing number of performance tests executed over last releases (see Fig. 2). Following the example of other organizations (Alsaqaf et al., 2019), the researcher proposed to reserve part of the sprint to non-functional assurance activities (such as software performance assessment), in order to mitigate the increasing prevalence of functional activities over non-functional ones. Nevertheless, the proposal was not accepted by the head of development and product owners.

The lack of continuous performance assessment was also faced. The head of development proposed to automate PR execution to enable continuous performance testing of PRs. A product owner and the *ChogoRi* scrum master raised several practical concerns about this proposal. According to them, the assessment of some PRs implied the configuration of complex environments, which can be difficult to automate. Moreover, these tests often required days for configuration (e.g., database population) and executions on multiple dedicated machines, which were often not available in the organization. The researcher proposed to classify PRs based on their complexity, and to execute lightweight and easily automatable tests more frequently (e.g., every night), and complex tests with minor frequency. Similarly, the quality assurance specialist highlighted the need of identifying a subset of automated test that could continuously provide performance feedback against software evolution.

Another point discussed was the role of the PR list in the organization. The head of the development and the agile coach perceived the PR list as one of the major source of troubles in the performance assessment process. According to the quality assurance specialist, many PRs were probably outdated and related to old versions of the product. During a meeting, the researcher showed that 35 over 92 PRs were never tested in the last 6 releases and many others were only tested few times (see Fig. 5), thereby raising the concerns of the head of development and product owners about the utility of some PRs. To deal with this problem, a revision of the PR list was proposed as a potential first step.

After these meetings, the head of development decided to prioritize two main activities for the improvement of the performance assessment process: (i) the identification of a subset of automated performance tests and (ii) a revision of the PR list.

In order to perform the first activity, a meeting was held involving all architects and product owners to identify a subset PRs for automation. Four main criteria were used to identify this subset: the relevance of system operation under test, the ease of automation, the time effort required to configure and run tests, and the robustness of tests in terms of maintenance. At the end of the process, 12 PRs that exercise architecturally relevant operations were selected, involving 7 web test PRs and 5 load tests PRs. UI and mixed tests were discarded due to their poor robustness in terms of maintenance.

The second activity was performed in a meeting involving all product owners and software architects. Architects were typically reluctant to remove PR, since they aimed to carefully assess every aspect of the performance of the system. However, many PRs were rarely tested due to time and resource constraints. During the process, architects were invited to reflect on the potential drawbacks, in terms of management, of having a large list of (rarely tested) PRs. Eventually, 5 PRs were deprecated in the revision process.

Summing up, several proposals were discussed to improve the performance assessment process. To enable an early performance feedback, two main proposals were discussed: the assignment of performance assessment activities to development teams and the automation of performance tests. The former was considered impractical due the potential overload of development teams and the lack of proper technical knowledge. The latter led to the identification of an automated performance test suite involving 12 architecturally relevant PRs. Another important problem that was discussed was the management of the PR list, which was considered a major source of troubles for the performance assessment process. To tackle this problem a revision of PR list was planned, which led to the deprecation of 5 PRs.

6 Findings

Through a qualitative analysis of the data collected during the ethnographic study, we distill three key challenges in improving the performance assessment process in ASD, which we discuss in the following.

Managing performance assessment activities. The organization struggled to design a well-defined process to manage performance assessment activities. Potential solutions, borrowed from current practices, were discussed to improve the management of performance assessment activities, such as the use of PRs as constraints of backlog items[12] , or the use of the Definition of Done[13] (Alsaqaf et al., 2019; Behutiye et al., 2020b). Nevertheless, these approaches were considered unsuitable for the organization. A challenge observed was the difference, in terms of goals and characteristics, of different performance assessment activities, which made difficult to design a unique process that works properly for all of them. For instance, some performance testing tasks were inherently linked to development activities (*e.g.,* assessment of new features), while others aimed to continuously monitor performance of architecturally relevant operations against system evolution. However, while the assessment of the latter was potentially crucial for any future software version, the relevance of the former was typically associated to a one-time development activity. Nevertheless, the organization managed both these performance activities in the same way, *i.e.,* they were both treated as permanent requirements. Therefore, a PR added with a high priority, for the assessment of a new feature, might remain highly prioritized over time due to the lack of priority updates (see Section 4.2), thereby leading to a suboptimal selection of performance assessment activities for future software releases.

Further studies are needed to design novel management approaches, which takes into account the differential nature of performance assessment activities.

Continuous performance assessment. Continuous assessment of software performance usually implies test automation. According to several informants within the organization, accurate performance tests are often difficult to automate due their effort in terms of time and resources. To tackle this problem, the case company organized a meeting with goal of selecting a subset of PRs for automation. Four main criteria were used to identify this subset: the relevance of the system operation under test, the ease of automation, the time effort required to configure and run tests, and the robustness of tests in terms of maintenance. At the end of this process, 12 architecturally relevant operations PRs were selected. The choice of proper tests suite for automation and proper frequency of execution is crucial for successful performance assessment. The main challenge, in this regard, is to provide an adequate tradeoff between accuracy and frequency. Indeed, accurate tests usually involves complex configurations of production-like environments and long execution times, hence they cannot be frequently executed. On the other, lightweight tests are easily automatable and can be executed more frequently, but they are less representative of the real system usage, hence they have lower chances of detecting performance bugs. The organization relied on the knowledge of software architects to identify a subset of automated tests. However, nowadays, there is

[12] Nonfunctional Requirements - Scaled Agile Framework. `https://bit.ly/3ohrZun`

[13] Definition of Done, Agile Alliance. `http://bit.ly/2YbdkWS`

still little knowledge on how to properly tackle tradeoff between accuracy and frequency in performance assessment. More empirical studies are needed to fill this gap; for example, it would be valuable to understand to what extent lightweight and easily automatable performance tests (*e.g.,* microbenchmarks (Laaber and Leitner, 2018)) can mitigate the risks of performance failures.

Performance testing automation also implies to consistently establish whether a test is passed or not (Fagerström et al., 2016). This problem was discussed in the organization, but no available solution was found. Although this problem was partially addressed through the use of targets in PRs, the identification of a performance bug often requires a thorough analysis of operational data that goes beyond the simple target check. The latter point was confirmed by two members of the *ChogoRi* team, which emphasized the need for a human manual analysis to reliably determine whether a test is passed or not. Additionally, many PRs did not involve a target (i.e., benchmarks), hence, in these cases, an automated technique is required to provide verdicts. The techniques proposed by Daly et al. (2020), Daly (2021), Nguyen et al. (2012) and Chen et al. (2017) seem promising in this regard.

Nevertheless, further research is needed on this topic.

Defining the performance assessment effort. Another key challenge is the definition of the proper work effort devoted to performance assessment activities. The definition of a clear and commonly accepted balance between performance assessment activities and other development activities is essential to enable a reliable performance assurance process. Indeed, different actors in the organization had different perceptions on the relevance of performance assessment. Hence, when defining the performance assessment effort, it is crucial to consider these different viewpoints. For example, during the PR revision, software architects considered every single PR essential to enable reliable performance assessment. On the other hand, the head of development and product owners tended to de-prioritize these types activities during software development, thereby allocating resources that were not sufficient to perform the amount of performance assessment activities expected by architects. These conflicting viewpoints and the lack of a commonly shared vision on the amount of effort devoted to performance assessment led to an increasing number of (rarely tested) PRs, and caused several issues to the performance assessment process. Further empirical studies are needed to investigate whether this is common problem in ASD, and to identify potential best practices.

7 Discussion

Throughout this section, we consider whether our findings may be applicable to other agile organizations, by comparing our key insights with those of previous studies reported in the literature.

The first key challenge we discovered concerns the *management of performance assessment activities.* Although prior work did not explicitly report this challenge for performance assurance in ASD, similar problems emerged in the (broader) context of non-functional quality assurance. Several studies reported difficulties in the management of Non-Functional Requirements (NFR) in ASD (Alsaqaf et al., 2019; Kasauli et al., 2021; Alsaqaf et al., 2017; Ramesh et al., 2010; Inayat et al., 2015; Behutiye et al., 2020a). For example, Alsaqaf et al. (2017) highlighted the inadequacy of user stories to document NFRs. In a subsequent study (Alsaqaf et al., 2019), the same authors reported that agile teams often rely on custum solutions to cope with this inadequacy (*e.g.,* assumption's wiki, multiple product backlogs, rules in monitoring tools). The lack of well-established guidelines for NFR management in ASD was also confirmed by the recent study of Behutiye et al. (2020b), which found that different practices are adopted to document NFRs (*e.g.,* user stories, Definition of Done, acceptance criteria, documents, artifacts, prototypes and also face-to-face communication). We observed a similar trend in our case study company, in which different teams used different methods to handle and document performance assessment activities (see Section 4.1.3). Another interesting fact reported in the literature is that there is often an unclear conceptual definition of NFR in ASD. According to Alsaqaf et al. (2019), practitioners differ in how they define the nature of NFR. For example, some consider NFRs as stand-alone requirements which should not be treated differently from other functional requirements, while others see NFRs as constraints over functional requirements. In our study, we observed a dichotomy in the nature of PRs. For example, some PRs were strictly tightened to functional development activities, while others were more similar to stand-alone requirements without specific relations with development activities. The inability to deal with these different kinds of PRs led to a suboptimal management of performance assessment activities. Another relevant problem concerns the lack of an explicit mechanisms for updating or changing requirements in agile frameworks, as reported by Kasauli et al. (2021). Interestingly enough, they reported that a case company was facing problems in keeping updated the list of NFRs. Originally, they planned to perform regularly updates on the NFRs list, however this approach only worked in the short term, subsequently the list started to become slowly out of date. We observed the same problem in our case organization as described in Section 4.2.1. In our case organization, the latter problem was strictly related to the lack of availability of relevant professional figures, such as product owners and architects. Interestingly, Alsaqaf et al. (2017) reported the heavy workload of product owners' and their insufficient availability as two main challenges for non-functional quality assurance in ASD.

Another key challenge we found is *continuous performance assessment.* In Section 4.2.2, we showed through an explanatory example the potential side-effects of the late identification of a performance issue. Empirical studies on non-functional quality assurance reported similar results. For example, Behutiye et al. (2020a) reported the late consideration of NFRs as a major challenge for quality assurance in ASD. According to them, handling non-functional issues in the late phases of development leads to unpredictable effort estimation and may induce the introduction of many changes. Alsaqaf et al.

(2017) also reported as a key challenge the validation of NFRs too late in the process. To tackle this problem, Johnson et al. (2007) proposed to incorporate performance testing in test-driven development through a technique called test-first performance (TFP), thus enabling early performance feedback from the system under evolution. According to them, the continuous performance feedback increased performance awareness during development and induced developers to design and code for better performance. During our study, the researcher proposed to assign performance assessment activities directly to development teams to enable early assessment of development activities. Nevertheless, the proposal was discarded by the management due to the potential overload for development teams and the lack of adequate technical skills. Interestingly, Behutiye et al. (2020a) reported the limited NFR expertise of agile teams as a key challenge for quality assurance in ASD. According to them, team members who lack NFR skills may emphasize implementation of functional requirements and ignore non-functional quality assurance.
Automated performance testing is also crucial to enable continuous performance feedback against the evolving system. Our case company faced two main problems when trying to automate performance tests: (i) the high effort (in terms of time and resources) required to perform accurate performance tests and (ii) the lack of clear "pass/fail verdicts" in performance tests. The relevance of these problems beyond the studied setting is confirmed by the recent interest of the research community in reducing the effort of performance tests (Laaber et al., 2020; Ding et al., 2020; Alghmadi et al., 2016) and providing automated verdicts (Fagerström et al., 2016; Daly et al., 2020; Chen et al., 2017).

The third challenge we discovered concerns the *definition of the performance assessment effort*. Several empirical studies reported that ASD often induces a reduction of the effort devoted to non-functional quality assurance. Behutiye et al. (2020a) reported that ASD management teams often tends to prioritize more feature development goals, and the focus on prioritizing only business value may lead to problems in terms non-functional quality. A similar tendency was observed by Alsaqaf et al. (2019), which reported that user stories priorities can be ignored when deadlines are approaching and the need to deliver functional features becomes greater than non-functional quality assurance. According to them, the commitment to deliver the software in time translates into focusing on functional development alone. Nevertheless, Cao and Ramesh (2008) reported that ASD teams that focused only on business value prioritization faced challenges in system efficiency and security, which affected the success of the software product. Kasauli et al. (2021) criticized widely popular ASD frameworks, such as LeSS and SAFe, for not providing any concrete guidance on how to balance the trade-off between product quality and time-to-market pressure. Indeed, companies usually adopts custom solutions to tackle this problem. For example, Alsaqaf et al. (2019) reported that one of their case companies reserved parts of the sprint to non-functional quality assurance activities such as performance assessment. Another case company instead used three different product backlogs to deal with different viewpoints of stakeholders. One of them was filled up with user stories by the product owner and represented the customer's business desires. While another was filled up by the software architect and represented non-functional quality assurance activities. In our study, we observed that a vague definition of the performance assessment effort and the lack of management of conflicting stakeholders' viewpoints induced a suboptimal performance assessment.

Overall, *the consistency of our insights with those of other studies suggests that our findings are relevant beyond the studied setting.*

Most of the prior work broadly focus on non-functional quality assurance, and do not examine how agile teams tackle non-functional quality assurance in their daily work. The few previous studies specifically addressing performance assurance in ASD only tackle specific aspects such as PR management (Chih-Wei Ho et al., 2006) or continuous performance assessment (Johnson et al., 2007). Through an ethnographic study, we investigated the whole performance assurance process in an ASD organization. The "thick picture" provided by ethnography enabled us to gather an holistic view on the performance assurance process, and to distill challenges involving both process (*e.g.,* management and documentation of performance assessment activities) and technical aspects (*e.g.,* performance testing automation). This novel holistic perspective can be useful to (i) software organizations, for avoiding the disclosed issues and leveraging the reported advices for the improvement of performance assurance in ASD, and (ii) researchers, for driving specific directions in software performance engineering research that are grounded in practice.

8 Threats to validity

8.1 Construct validity

Research involving people observation, such as ethnographic studies, may originate issues in terms of bias and rigor. Empirical research in industrial practice puts the researcher in a situation influenced by contradicting interests, hierarchies, and personal antagonisms (Dittrich, 2002). To mitigate such issues, we first sought to establish a prolonged involvement in the fieldwork by keeping close contact with the organization members for six months. The development of a trusting relationship helped us to collect data from different perspectives, and also to observe the organization actors working in different Scrum ceremonies, meetings and daily activities. We also had access to system and process documentation. The significant amount of gathered data supported data triangulation which gave us a better confidence in our interpretation. In order to guarantee the quality of our descriptions, we took into account data gathered from different sources, *e.g.,* photographs, digital copies of documents and hand-written notes.

Following the best practices in ethnographic research (Sharp et al., 2016), we didn't rely on a rigid plan, instead we keep data collection plans and expectations flexible to keep an "open mind" about the practices under study. Although this flexibility may suggest a lack of rigor, it is also considered as one of the main strengths of ethnographic research (Fetterman, 2019).

8.2 Internal validity

In our study, we were interested in understanding the practices of performance assurance as practitioners were interested in improving such practices. This might have influenced the participants' interaction with the researcher, as practitioners often expect recommendations and improvements when engaging with researchers conducting an ethnographic study (Sharp et al., 2016). Indeed, the researcher could be perceived as a managerial agent who will provide recommendations to improve their process. On the other hand, practitioners might hide aspects of their practice they do not want to have documented for management. In order to minimize this threat, we tried to be explicit about the intention of our research and the interaction with the involved stakeholders.

In the last three months of the study, the researcher actively participated to the improvement of the performance assurance process, thereby potentially influencing the outcome of the research. Although this could be a potential threat, the intent of our ethnographic study is to provide not only an in-depth understanding of the performance assurance practices, but also to support the improvement of such practices. Indeed, simply understanding practice may not be enough to satisfy the purpose of our research, as we aim to also understand the challenges in improving the performance assurance process. Additionally, when performing an ethnographic study, practitioners typically expect help by the researcher with improving their practices, and they could be puzzled if the researcher does not help to improve them (Sharp et al., 2016).

The results obtained with ethnography tends to have a high internal validity (Sharp et al., 2016) as far as the situation or context within which the evidence is gathered is consistent with the aim of the study (McGrath, 1995). Indeed, we conducted an ethnographic study in a R&D division of a large company that adopts Scrum (Rubin, 2012), *i.e.,* a widely popular Agile development process, as we aim to understand the practices and the challenges for software performance assurance in the context of ASD.

8.3 External validity

There could be a limitation by focusing on one organization. Indeed, ethnographic studies are often criticized to have a weak external validity, *i.e.,* do not necessarily generalize to other contexts (McGrath, 1995). For example, the practices adopted by other organizations may be different and more effective. However, we compared the problems and challenges found in our study with those reported by other non-functional quality assurance studies in Agile contexts (*e.g.,* (Alsaqaf et al., 2019; Behutiye et al., 2020a; Kasauli et al., 2021; Behutiye et al., 2020b)). The consistency of our results with the ones presented in other studies gives us confidence that the main findings of this study are portable to other organizations.

9 Related work

There are few studies concerning performance assurance in the context of ASD. Chih-Wei Ho et al. (2006) proposed an evolutionary model for PRs specifications, called PREM. PREM provides guidelines on the level of detail needed in a PR for development teams to specify the necessary performance details and the form of validation for the PR. Our case organization used an approach somehow similar to PREM, where PRs were first roughly defined by product owners (or architects), and then iteratively refined with the testing team. The same authors of PREM, in another article (Johnson et al., 2007), described an experience in incorporating performance tests in Test-Driven Development. In our study, the researcher proposed to integrate performance testing activities as part of the development cycle. Nevertheless, the proposal was rejected since development teams were considered too busy and unexperienced to deal with this kind of activities.

Studies investigating non-functional quality assurance in the context ASD are related to our work. In an early study, Ramesh et al. (2010) identified inadequate attention given to non-functional requirements as a major issue in ASD, by reporting specific concerns on software performance. The same problem was also reported in a systematic literature review on requirement engineering practices in ASD (Inayat et al., 2015). Similarly, in a recent empirical study, Kasauli et al. (2021) reported difficulty in handling NFRs in the context of large-scale ASD, highlighting a lack of proper solutions to effectively handle NFRs. Interestingly, they reported that, similarly to our case organization, one of their case companies was facing problems in handling and keeping updated the list of NFRs. On the other hand, they also reported a reluctance to record NFRs in Agile companies, while we didn't observe a similar behavior in our case organization.

Behutiye et al. (2020a) identified four top categories of challenges for NFR management in ASD: the limited ability of ASD to handle NFR, time constraints due to short iteration cycles, limitations regarding the testing of NFRs and neglect of NFRs were the top categories of challenges. According to them, short iteration cycles and time constraints minimize the focus on addressing NFRs, and emphasize more implementation of functional features. In another study, Behutiye et al. (2020b) investigated documentation of NFRs in ASD. They found that different practices are used to document NFRs: user stories, Definition of Done, acceptance criteria, documents, artifacts, prototypes and also face-to-face communication. According to them, companies approach to documentation of NFRs to fit the needs of their context, and the choice of the practices is affected by factors such as product domain, organization size and practitioners' experience.

Through a systematic literature review, Alsaqaf et al. (2017) identified challenges that harm the management of NFRs in large-scale distributed Agile projects. Among them, there are: the inability of user stories to document NFRs, and the validation of NFRs that occurs too late in the process. The same researchers performed an empirical study (Alsaqaf et al., 2019) to identify mechanisms behind the challenges of managing NFRs in large-scale distributed Agile projects, and practices used to mitigate the impact of these challenges. They found that minimal documentation might result in missing the rationale behind NFR tradeoffs and architecture decisions taken earlier. Moreover, according to their findings, the priorities associated with user stories could be shuffled when deadlines are approaching and the need to deliver functional features becomes more critical than verifying NFRs. Example of practices adopted to mitigate challenges involves: reserving part of the sprint for NFRs assessment and the use of multiple product backlogs to include requirements of different viewpoints. Other studies address quality assurance in ASD in the specific context of safety-critical systems (Hanssen et al., 2016; Fitzgerald et al., 2013).

Prior work broadly investigate non-functional quality assurance in ASD, while our study specifically target performance assurance. Moreover, these studies present results based on data collected through interviews and surveys, but do not examine how agile teams tackle non-functional quality assurance in their daily work. Instead, we explored how performance assurance is integrated in software development daily work by means of an ethnographic qualitative approach (Sharp et al., 2016).

10 Conclusion

Our research objective was to explore practice and challenges for performance assurance in ASD. Through our ethnographic study, we showed that companies face several challenges in ensuring software performance in ASD. The lack of guidelines and well-established practices induces the adoption of approaches that can be obsolete and inadequate for the observed contexts, thereby leading to sub-optimal management of performance assurance activities.

An important concern is the definition of the proper effort devoted to performance assessment activities. Indeed, different actors of an organization (*e.g.,* product owners and software architects) have different perceptions on the relevance of performance assessment activities. Hence, a clear and commonly shared definition of the amount of effort devoted to these activities is crucial to enable a reliable performance assurance process.

Continuous performance assessment also plays a relevant role in ASD. Indeed, the late identification of performance bugs may imply time-consuming debugging, expensive reworks and potential technical debt. In that, performance testing automation becomes essential to enable continuous performance feedback against frequent code changes. A main challenge, in this regard, is the identification of a proper tradeoff between test frequency and accuracy. Indeed, accurate performance tests cannot be frequently executed, since they require complex production-like environments and high amount of hardware resources. On the other hand, lightweight performance tests are easier to automate and less demanding in terms of resources, but they are also less representative of the real system usage, hence less prone to discover performance bugs. More empirical studies are needed to understand to what extent lightweight performance tests (*e.g.,* microbenchmarks (Laaber and Leitner, 2018)) can mitigate the risks of major performance failures. Another relevant challenge for the automation of performance tests is the lack of a clear pass/fail verdict. Performance test results (*e.g.,* mean execution time) are usually compared to a baseline (*e.g.,* results of a previous software version) to determine a verdict, and it is often difficult to judge (in an automated way) whether the performance change is relevant or not. State-of-the-art approaches leverage change point detection (Daly et al., 2020), statistical process control techniques (Nguyen et al., 2012) or regression models (Chen et al., 2017) to identify significant changes from the history of performance results. Nevertheless, further studies are needed to design reliable automated verdicts for continuous performance assessment.

In summary, this paper has shown that the assessment of software performance in the context of ASD is still far from being flawless. Further research is needed to improve the management of performance assessment activities, and to enable effective continuous performance assessment. In particular, we encourage future studies to investigate performance assurance practices and challenges in other Agile organizations with different characteristics (*e.g.,* domain, organization size, development methodology) to further strengthen (or debunk) the validity of our findings.

Acknowledgements This work is partially founded by Territori Aperti (a project funded by Fondo Territori Lavoro e Conoscenza CGIL, CSIL and UIL) and by the project "Software Performance in Agile/DevOps context" funded within Programma Operativo Nazionale Ricerca e Innovazione 2014-2020. I would like to thank Vittorio Cortellessa for making this industry collaboration possible. I would also like to thank him for the useful suggestions and comments that were helpful in improving the paper.

References

Alghmadi HM, Syer MD, Shang W, Hassan AE (2016) An automated approach for recommending when to stop performance tests. In: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp 279–289, DOI 10.1109/ICSME.2016.46

Alsaqaf W, Daneva M, Wieringa R (2017) Quality requirements in large-scale distributed agile projects – a systematic literature review. In: Grünbacher P, Perini A (eds) Requirements Engineering: Foundation for Software Quality, Springer International Publishing, Cham, pp 219–234

Alsaqaf W, Daneva M, Wieringa R (2019) Quality requirements challenges in the context of large-scale distributed agile: An empirical study. Information and Software Technology 110:39 – 55, DOI https://doi.org/10.1016/j.infsof.2019.01.009, URL http://www.sciencedirect.com/science/article/pii/S0950584918300739

Anderson B (1997) Work, ethnography and system design. In: Kent A, Williams JG (eds) The Encyclopedia of Microcomputers, vol 20, Marcel Dekker, New York, NY, USA, pp 159–183

Auer K, Beck K (1996) Lazy Optimization: Patterns for Efficient Smalltalk Programming, Addison-Wesley Longman Publishing Co., Inc., USA, pp 19–42

Beck K, Beedle M, van Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R, Kern J, Marick B, Martin RC, Mellor S, Schwaber K, Sutherland J, Thomas D (2001) Manifesto for agile software development. URL http://www.agilemanifesto.org/

Behutiye W, Karhapää P, López L, Burgués X, Martínez-Fernández S, Vollmer AM, Rodríguez P, Franch X, Oivo M (2020a) Management of quality requirements in agile and rapid software development: A systematic mapping study. Information and Software Technology 123:106225, DOI https://doi.org/10.1016/j.infsof.2019.106225, URL http://www.sciencedirect.com/science/article/pii/S095058491930240X

Behutiye W, Seppänen P, Rodríguez P, Oivo M (2020b) Documentation of quality requirements in agile software development. In: Proceedings of the Evaluation and Assessment in Software Engineering, Association for Computing Machinery, New York, NY, USA, EASE '20, pp 250–259, DOI 10.1145/3383219.3383245, URL https://doi.org/10.1145/3383219.3383245

Brutlag J (2009) Speed matters. URL http://services.google.com/fh/files/blogs/google_delayexp.pdf, [online] http://services.google.com/fh/files/blogs/google_delayexp.pdf

Cao L, Ramesh B (2008) Agile requirements engineering practices: An empirical study. IEEE Software 25(1):60–67, DOI 10.1109/MS.2008.1

Chen TH, Syer MD, Shang W, Jiang ZM, Hassan AE, Nasser M, Flora P (2017) Analytics-driven load testing: An industrial experience report on load testing of large-scale systems. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), pp 243–252, DOI 10.1109/ICSE-SEIP.2017.26

Chih-Wei Ho, Johnson MJ, Williams L, Maximilien EM (2006) On agile performance requirements specification and testing. In: AGILE 2006 (AGILE'06), pp 6 pp.–52, DOI 10.1109/AGILE.2006.41

Daly D (2021) Creating a virtuous cycle in performance testing at mongodb. In: Proceedings of the ACM/SPEC International Conference on Performance Engineering, Association for Computing Machinery, New York, NY, USA, ICPE '21, p 33–41, DOI 10.1145/3427921.3450234, URL https://doi.org/10.1145/3427921.3450234

Daly D, Brown W, Ingo H, O'Leary J, Bradford D (2020) The use of change point detection to identify software performance regressions in a continuous integration system. In: Proceedings of the ACM/SPEC International Conference on Performance Engineering, Association for Computing Machinery, New York, NY, USA, ICPE '20, pp 67–75, DOI 10.1145/3358960.3375791, URL https://doi.org/10.1145/3358960.3375791

Ding Z, Chen J, Shang W (2020) Towards the use of the readily available tests from the release pipeline as performance tests: Are we there yet? In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Association for Computing Machinery, New York, NY, USA, ICSE '20, p 1435–1446, DOI 10.1145/3377811.3380351, URL https://doi.org/10.1145/3377811.3380351

Dittrich Y (2002) Doing empirical research on software development: Finding a path between understanding, intervention, and method development. In: Social Thinking-Software Practice, The MIT Press, pp 243–262

Fagerström M, Ismail EE, Liebel G, Guliani R, Larsson F, Nordling K, Knauss E, Pelliccione P (2016) Verdict machinery: On the need to automatically make sense of test results. In: Proceedings of the 25th International Symposium on Software Testing and Analysis, Association for Computing Machinery, New York, NY, USA, pp 225–234, DOI 10.1145/2931037.2931064, URL https://doi.org/10.1145/2931037.2931064

Fetterman DM (2019) Ethnography: Step-by-step. Sage Publications

Fitzgerald B, Stol K, O'Sullivan R, O'Brien D (2013) Scaling agile methods to regulated environments: An industry case study. In: 2013 35th International Conference on Software Engineering (ICSE), pp 863–872, DOI 10.1109/ICSE.2013.6606635

Fowler M (2002) Yet another optimisation article. IEEE Software 19(3):20–21, DOI 10.1109/MS.2002.1003448

Geertz C (1988) Works and lives : the anthropologist as author. Stanford University Press, Stanford, Calif.

Hanssen GK, Haugset B, Stålhane T, Myklebust T, Kulbrandstad I (2016) Quality assurance in scrum applied to safety critical software. In: Sharp H, Hall T (eds) Agile Processes, in Software Engineering, and Extreme Programming, Springer International Publishing, Cham, pp 92–103

Heikkila VT, Paasivaara M, Lassenius C (2013) Scrumbut, but does it matter? a mixed-method study of the planning process of a multi-team scrum organization. In: 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, pp 85–94, DOI 10.1109/ESEM.2013.27

Inayat I, Salim SS, Marczak S, Daneva M, Shamshirband S (2015) A systematic literature review on agile requirements engineering practices and challenges. Computers in Human Behavior 51:915 – 929, DOI https://doi.org/10.1016/j.chb.2014.10.046, URL http://www.sciencedirect.com/science/article/pii/S074756321400569X, computing for Human Learning, Behaviour and Collaboration in the Social and Mobile Networks Era

Jiang ZM, Hassan AE (2015) A survey on load testing of large-scale software systems. IEEE Transactions on Software Engineering 41(11):1091–1118, DOI 10.1109/TSE.2015.2445340

Johnson MJ, Ho C, Maximilien EM, Williams L (2007) Incorporating performance testing in test-driven development. IEEE Software 24(3):67–73, DOI 10.1109/MS.2007.77

Kasauli R, Knauss E, Horkoff J, Liebel G, de Oliveira Neto FG (2021) Requirements engineering challenges and practices in large-scale agile system development. Journal of Systems and Software 172:110851, DOI https://doi.org/10.1016/j.jss.2020.110851, URL http://www.sciencedirect.com/science/article/pii/S0164121220302417

Knuth DE (2007) Computer Programming as an Art, Association for Computing Machinery, New York, NY, USA, p 1974. URL https://doi.org/10.1145/1283920.1283929

Laaber C, Leitner P (2018) An evaluation of open-source software microbenchmark suites for continuous performance assessment. In: Proceedings of the 15th International Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA, MSR '18, pp 119–130, DOI 10.1145/3196398.3196407, URL https://doi.org/10.1145/3196398.3196407

Laaber C, Würsten S, Gall HC, Leitner P (2020) Dynamically reconfiguring software microbenchmarks: reducing execution time without sacrificing result quality. In: Devanbu P, Cohen MB, Zimmermann T (eds) ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020, ACM, pp 989–1001, DOI 10.1145/3368089.3409683, URL 10.1145/3368089.3409683

McGrath JE (1995) Methodology Matters: Doing Research in the Behavioral and Social Sciences, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p 152–169

Meyer H (2009) Manufacturing Execution Systems: Optimal Design, Planning, and Deployment. McGraw-Hill Education, New York, URL https://www.accessengineeringlibrary.com/content/book/9780071623834

Nguyen TH, Adams B, Jiang ZM, Hassan AE, Nasser M, Flora P (2012) Automated detection of performance regressions using statistical process control techniques. In: Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, Association for Computing Machinery, New York, NY, USA, ICPE '12, p 299–310, DOI 10.1145/2188286.2188344, URL https://doi.org/10.1145/2188286.2188344

Ramesh B, Cao L, Baskerville RL (2010) Agile requirements engineering practices and challenges: an empirical study. Inf Syst J 20(5):449–480, DOI 10.1111/j.1365-2575.2007.00259.x, URL https://doi.org/10.1111/j.1365-2575.2007.00259.x

Rubin KS (2012) Essential Scrum: A Practical Guide to the Most Popular Agile Process, 1st edn. Addison-Wesley Professional

Sharp H, Dittrich Y, de Souza CRB (2016) The role of ethnographic studies in empirical software engineering. IEEE Transactions on Software Engineering 42(8):786–804, DOI 10.1109/TSE.2016.2519887

Smith C, Williams L (2001) Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software. Addison-Wesley object technology series, Addison-Wesley, URL https://books.google.it/books?id=X5VlQgAACAAJ

Woodside M, Franks G, Petriu DC (2007) The future of software performance engineering. In: 2007 Future of Software Engineering, IEEE Computer Society, USA, FOSE '07, pp 171–187, DOI 10.1109/FOSE.2007.32, URL https://doi.org/10.1109/FOSE.2007.32