Continuously Managing NFRs: Opportunities and Challenges in Practice

Colin Werner, Ze Shi Li, Derek Lowlind, Omar Elazhary, Neil Ernst, and Daniela Damian

Abstract—Non-functional requirements (NFR), which include performance, availability, and maintainability, are vitally important to overall software quality. However, research has shown NFRs are, in practice, poorly defined and difficult to verify. Continuous software engineering practices, which extend agile practices, emphasize fast paced, automated, and rapid release of software that poses additional challenges to handling NFRs. In this multi-case study we empirically investigated how three organizations, for which NFRs are paramount to their business survival, manage NFRs in their continuous practices. We describe four practices these companies use to manage NFRs, such as offloading NFRs to cloud providers or the use of metrics and continuous monitoring, both of which enable almost real-time feedback on managing the NFRs. However, managing NFRs comes at a cost—as we also identified a number of challenges these organizations face while managing NFRs in their continuous software engineering practices. For example, the organizations in our study were able to realize an NFR by strategically and heavily investing in configuration management and infrastructure as code, in order to offload the responsibility of NFRs; however, this offloading implied potential loss of control. Our discussion and key research implications show the opportunities, trade-offs, and importance of the unique give-and-take relationship between continuous software engineering and NFRs.
Research artifacts may be found at https://doi.org/10.5281/zenodo.3376342.

Index Terms—non-functional requirements, continuous software engineering

1 Introduction

Non-functional requirements (NFRs), also known as quality attribute requirements, represent attributes or constraints on a system [1]. NFRs are very important in software projects, greatly influencing underlying software architecture [2]. For many software organizations, particularly in today's increasingly service-oriented, fast paced software development marketplace, NFRs such as system uptime, code maintainability, and responsiveness, are vital to success [3]. For example, a recent software outage at Amazon was estimated to cost 99 million USD for the 63 minutes Amazon's site was down [4]. The majority of software organizations [5] now use continuous software engineering (CSE) [6], [7] which involves rapid and frequent builds, automated tests, as well as transparency of the build and verification process [6].

Despite the great benefits organizations reap through CSE [8], NFRs have rarely been explored in the context of CSE [9]. Studies of practice demonstrate NFRs are inherently difficult to explicitly express [10] and even more difficult to verify or validate [11], whether they are part of a formal requirements specification or an agile user story. In agile settings NFRs present more engineering difficulties than functional requirements [12]. Continuous settings that focus on automated and rapid release of software compound these challenges, for example in A/B testing [13]. This leads to long-term sustainability issues and mounting technical debt [14] resulting in costly and unnecessary rework [15].

Fowler's original definition of CSE practices [6] only mentions testing of NFRs. In addition to testing, however, managing, modeling, and eliciting NFRs, or *quality* requirements, are all central to CSE's overall goal of improving software quality [16]. The software engineering literature lacks empirical evidence on how organizations can verify, let alone manage, implement, and realize an NFR by leveraging CSE [17].

In this paper we report on an exploratory study that aims to fill this gap through empirical insights on the relationship between NFRs and CSE, from an industrial multiple-case study [18] of managing NFRs in CSE. Two research questions guided our study:

RQ1 How do CSE organizations manage NFRs?
RQ2 What challenges does CSE introduce when managing NFRs?

We investigate the CSE practices at three organizations for which NFRs, such as performance, security and availability, are paramount to their business survival. We collected data by observing employees in a number of immersive visits we conducted at these organizations, and through 18 semi-structured follow-up interviews. We interacted with a variety of roles that are directly involved with or impacted by how NFRs were handled in CSE, including developers, DevOps engineers, QA testers, and managers. Following thematic analysis [19] on our rich qualitative data, we identify a set of practices that these organizations employ to handle their NFRs (RQ1), e.g. letting someone else manage an NFR by offloading it to a third-party. The answer to RQ2 exposes the challenges CSE introduces when managing NFRs, e.g. the fast pace of development led to a decrease in the shared understanding of NFRs.

The contributions of this paper include empirical evidence of the practices and challenges faced by organizations dealing with NFRs in CSE, and the implications we draw for research and industry. In addition, we bring awareness to academics and practitioners on the unique relationship,

opportunities, and trade-offs that CSE offers to NFRs. In particular,

- how organizations using CSE can manage NFRs, for example by offloading sub-tasks of NFRs to third-parties,
- managing an NFR comes at the cost of certain trade-offs, such as the loss of control over an offloaded NFR, or a decrease in the shared understanding of the NFR, and
- the importance of CONFIGURABILITY as an NFR and the amount of investment required to manage it.

We consider our empirically-derived insights to be useful hypotheses for future research to validate in more organizations that practice CSE. The recent systematic literature review (SLR) by Yu et al. [17] on the state-of-the-art (and practice) of leveraging continuous integration[1] (CI) for NFR testing found that CI does indeed support the testing of some NFRs, while highlighting the very low ratio of industrial empirical evidence to theoretical research in this area. Our findings bring empirical evidence on a broader set of practices in managing NFRs (including testing) in CSE practices.

In the remainder of the paper, we first introduce the related literature on NFRs and requirements engineering in agile environments, and on NFRs in CSE in particular. We then describe our empirical research methodology. We introduce our findings in the form of practices and challenges we identified at the three organizations we studied. Finally, our discussion of these findings debates the opportunities, but also the trade-offs associated with managing NFRs in organizations practicing CSE.

2 Background and Related Work

Research into the ways in which CSE deals with NFRs is limited. We have a reasonably clear understanding of NFRs in general, and how agile software development and requirements engineering affect one another. However, these studies tend to be focused on larger contexts (e.g., in distributed teams or multi-team agile initiatives [12]), and make little mention of CSE. In particular, the relatively recent rise of CSE [7], such as automated verification and build-on-commit, has the potential to greatly impact how NFRs are managed, because they insist on automated verification, rapid iteration, and shared codebases. We discuss related work in the areas of requirements engineering (RE) in agile settings, including NFRs, and finally, work on NFRs and CSE.

2.1 Requirements Engineering and Agile

It is now accepted that RE in agile organizations follows a just-in-time requirements engineering approach [20], [21]. Just-in-time RE practices deal with requirements as needed, rather than upfront, for example, by adding issues to the backlog and then delving into the requirements for that issue only once (or if) it becomes part of the iteration plan [22]. Frequently members of an organization's leadership team are the only people with detailed knowledge of the requirements [20], which aligns with philosophies such as Ries' lean startup approach [23]. The focus is to release often, gather feedback on the new features delivered, and prioritize work for the next release as needed. The implication of this just-in-time RE is that a) little upfront analysis is done and b) requirements analysis is taking place in the verification and experimentation phases, usually once software is released to customers. This has implications for how an organization is measuring and analyzing its requirements. Agile RE risks neglect of NFRs, since user stories focus primarily on features [24]. Given that NFRs are difficult to analyze and understand, even in highly planned, up-front RE processes, suggests an even bigger challenge in just-in-time settings.

For the purposes of our study we use Martin Glinz's definition of an NFR, which is "an attribute of, or a constraint on, a system" [1]. On the surface, NFRs are often simplified as system qualities, such as the 'ilities': usability, reliability, maintainability, etc.; however, upon deeper analysis NFRs may have significant influence on a system's overall design and architecture [25]. NFRs tend to receive a lot of attention in safety-critical systems, or in large organizations. However, for small organizations in web application settings, including the organizations we studied, NFRs such as software reliability or system performance are also critically important (as we will show; see Table 5).

Despite this importance, NFRs for organizations in agile settings are often "informally stated, contradictory, difficult to enforce during development, and very hard to validate" [26]. For example, 75% of NFRs in a recent study from Eckhardt, Vogelsang, and Mendez [10] were actually describing system behaviour . Finally, the wide-ranging and extensive NaPiRE study [27] found that "unclear / unmeasurable non-functional requirements" were one of the top problems respondents had with requirements in their small organizations [28, p.11].

Moreover, *how* and *who* verifies NFRs are another important aspect of NFRs. Previous work found that NFRs are often difficult to verify and validate [11], [26]. Manual verification is often the most common choice to verify NFRs [3]. The study by Ramesh, Cao and, Baskerville [29] suggested that in agile RE, NFRs often get de-prioritized in small organization settings and NFRs are ill-defined and ignored, e.g. "We have no specific test of stability. We just test for functionality and see if it stays up" [29]. This makes dealing with NFRs at a later date more difficult, and it typically introduces technical debt.

Most recently, research has examined how best to incorporate NFRs into agile settings. For example Alsaqaf, Daneva, and Wieringa [12] examined the way large, multi-team projects managed NFRs. The main challenge was the way NFRs cross-cut teams. Their research however studied organizations that are not operating in CSE, and which is the focus of the research we report in this paper.

2.2 Non-Functional Requirements and Continuous Software Engineering

While there are other definitions of CSE, in this paper we use Martin Fowler's influential blog post on the topic [6]. Fowler's definition of CSE includes maintaining a single source code repository, automated verification, automated

1. Continuous integration is one aspect of continuous software engineering.

builds, fast builds, and automated deployment [6]. Fowler also specifies sound organizational practices, which include each developer creating a commit at least once a day and keeping the build process transparent for all stakeholders [6]. CSE is a paradigm that emphasizes rapid and automated releases of working software [16].

Research into the treatment of NFRs in CSE is limited. Similar to agile, it is mostly driven by FRs, with the primary focus of delivering functionality to end-users as early as possible to receive quick feedback [30], often at the expense of other aspects such as NFRs. While CSE has been shown to enhance requirement traceability [31], the majority of research that investigates NFRs in CSE mostly focuses on verification of NFRs. Although, NFRs are often neither comprehensively verified nor automated.

One study on verification found that an organization may not provide sufficient time to verify NFRs [32] because NFRs may require more time to verify. Late verification of NFRs may cause severe side effects, such as re-factoring architecture at a stage when an organization is busy preparing for release of a software [33]. Even when organizations do verify NFRs, the amount of automated tests for NFRs is limited, and an organization may require manual verification to verify them [34].

One aspect often missing in those prior works is description of tools used for NFRs. For example, Savor et al. [35] mention that NFRs were watched by measurement tools, but makes no mention of what tools or how the tools were operated. The recent SLR by Yu et al. [17] found that CI environments (and tools) could be leveraged to adequately verify NFRs; however, they are underutilized. Furthermore, there is a very low ratio of industrial studies in NFR verification in CI compared to academic studies [17].

While this previous research indicates that CI may be leveraged to *verify* an NFR [17], verifying an NFR is only one important aspect of NFRs. In particular, an organization may attempt to verify an important NFR to determine whether that NFR may or may not be realized, and perhaps to what degree. However, verifying an NFR does not help actually realize the NFR itself.

NFRs or quality attributes are typically decomposed into smaller units, either quality attribute scenarios [36] or tasks [37]. Realizing the NFR as a whole (or 'satisficing' [38]) requires verifying that each of these smaller units is achieved. Hence an NFR may be only partially realized, as any number of outstanding (or unknown) tasks may remain to fully realize (satisfice) an NFR. The type and number of these smaller units is context-dependent, as is the importance of each NFR. For example, one of our companies prioritized SCALABILITY, which they realized in part by offloading reliability tasks onto a cloud provider [39]. It remains unclear exactly *how* CSE can help an organization realize an NFR. Our study attempts to fill this gap by describing how organizations can manage and realize NFRs when using CSE.

3 Research Method

We use an exploratory approach in a multiple-case study [18] to investigate the practices in management of NFRs at three software development organizations using CSE. A case study methodology is the most appropriate research methodology when studying a contemporary phenomena in a real-life context, such as we are [18]. Given the intricate nature of this understudied research domain we employed a qualitative research methodology, which is suitable to study non-technical aspects, including socio-technical, that complement traditional quantitative software engineering research methods [40] and to develop empirically-driven theories in software engineering [41]. Our methodology is summarized in Figure 1.

TABLE 1
Participants and their Roles at the three studied organizations

Org.	P#	Role	Gender	Exp. at Org.	Overall Exp.
Alpha	P1	Dev.	Male	< 2y	< 20y
	P2	Dev.	Male	< 10y	< 20y
	P3	Mgr.	Male	< 10y	< 20y
	P4	Mgr.	Male	< 5y	< 10y
	P5	Mgr.	Male	< 10y	< 20y
Beta	P6	Dev.	Male	< 2y	< 20y
	P7	Mgr.	Female	< 5y	< 20y
	P8	Mgr.	Female	< 10y	< 10y
	P9	Dev.	Male	< 5y	< 5y
	P10	Dev.	Male	< 5y	< 20y
	P11	Dev.	Male	< 2y	< 20y
	P12	Dev.	Female	< 2y	< 2y
	P13	Dev.	Male	< 2y	< 5y
Gamma	P14	Dev.	Male	< 2y	< 2y
	P15	Mgr.	Female	< 2y	< 20y
	P16	Dev.	Male	< 2y	< 5y
	P17	Dev.	Male	< 5y	> 20y
	P18	Dev.	Female	< 2y	< 5y

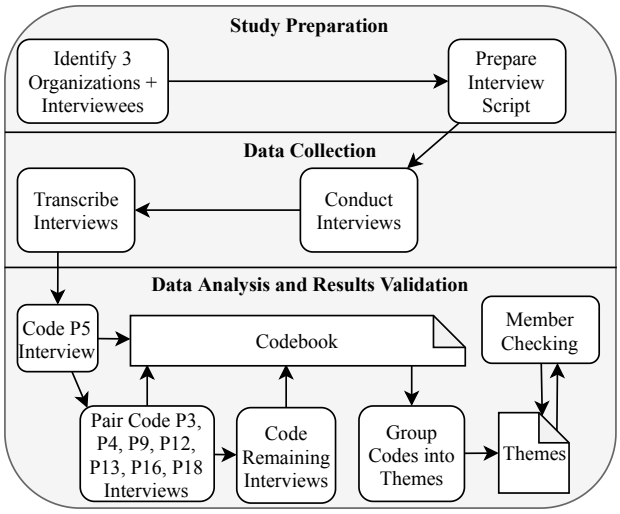

Fig. 1. Research Process

To recruit prospective organizations we used personal contacts at organizations using CSE, which resulted in identifying seven local organizations. We approached each organization to gauge their willingness to participate in our study and to understand the extent to which they manage NFRs and practice CSE in their development. Our prospective organizations covered a broad range of software domains (e.g. e-commerce, content management), number of employees, age of organization, and maturity of CSE practices. We selected three organizations based on avail-

ability and willingness to grant us research access, while each organization represented a different software domain and exhibited mature CSE practices for our study.

In particular, each organization has grown from a small startup into a mature, established leader in their respective business domains, and is implementing the recommended CSE best-practices to a high degree. In addition, in a subsequent survey with the study participants, we found 92% of respondents thought their respective organization manages NFRs well. In line with the tenets of case-study research, these organizations offered the opportunity to study the relationship between CSE and NFRs due to their realizing the importance of key NFRs and appropriately, and continuously, managing these NFRs early-on and throughout the organization's lifespan. Each of these organizations are 8 years old, have between 30-60 employees, and use some form of cloud provider (e.g Amazon Web Services, Google Cloud). Alpha, Beta, and Gamma all run automated builds, use automated tests, and use automatic deployments to production.

Alpha works in the data collection and analytics industry, processing large amounts of data on a daily basis. Beta provides an e-commerce platform with multiple platforms for customers distributed worldwide. Gamma is a content provider, including online advertisement management. We chose these three organizations (Alpha, Beta, and Gamma) because our intent was to study how *commercial organizations* perceived and managed NFRs in the context of CSE. As part of our approved research protocol and our NDAs, we anonymized the names of the organizations and interviewees.

3.1 Preliminary Study

We first sought to build our understanding by familiarizing ourselves with the personnel and the particular software development practices at each of the three organizations. One author spent multiple days over a few months embedded at each organization, learning about their products, processes, and lines of business [42]. While we did not have access to every single employee, through our immersive visits we were able to speak with a broad range of employees, spanning multiple organization boundaries. In particular, we spoke with 37 different employees (representing roughly $\frac{1}{3}$ of total staff), including 17 developers, 8 development managers, 5 product managers, 4 executives, 2 customer success specialists, 1 quality assurance member, and 1 director of sales. Thus we had access to a sub-set of employees, but included at least one employee from every major team. Our early immersive meetings and observations at our organizations informed our focused investigation into the lack of shared understanding of NFRs and its relationship to rework.

3.2 Data Collection

Once we had confidence in our understanding of each organization, we collected qualitative data through interviews of eighteen employees in development and managerial roles from these organizations A summary of the interviewees is in Table 1. The semi-structured, open-ended interviews were conducted by two authors with each participant and lasted between 45-90 minutes in person at an organization's office. We created a template of fourteen base interview questions for our semi-structured interviews[2]. Our interviews included questions on NFRs and CSE, such as: 1) organization definition of an NFR (e.g. *Do you define NFRs? How do you define an NFR? Which NFRs are important to you organization?*) 2) organization definition and implementation of CSE (e.g. *Are you familiar with the term continuous integration, delivery, and deployment? If familiar, how do you define those terms? Does your organization practice CSE?*), as well as about how their organization managed NFRs: 3) treatment of NFRs in context of CSE (e.g. *How do you trace an NFR through the continuous deployment? What happens when an NFR fails; is there a feedback loop from continuous development?*).

We started each interview by going through our base set of questions. We followed with additional probing questions on more specific subjects, depending on an interviewee's role or primary work. For example, an interview with a DevOps engineer involved extended questioning on tools or processes used by the engineer's organization to facilitate their CSE. An interview with a front-end developer working extensively on the visual aesthetics of an application often included questions regarding prioritizing and verification of NFRs. Interviews were, with permission, recorded.

3.3 Data Analysis and Results Validation

We transcribed each recorded interview using an automated transcription service, and verified each transcription by a human. Subsequently, we employed thematic analysis, an established data analysis method to identify themes and patterns in our data [19]. Our analysis involved inductively developing codes from the raw transcripts and then identifying themes that related to practices and challenges of handling NFRs in our studied organizations. The open coding approach [43] was used to minimize the bias any one particular coder could have. Throughout our coding we used the constant comparison method, whereby codes were added, removed, and merged based on the discussions between the coders.

After each coding session all coders would meet to discuss each item, which codes were applied, and debate the reasoning behind a particular code and whether that code actually applied to that item. In our initial coding phase, the first four authors coded every dialog segment of the same transcript (P5) (a 'dialog segment' was our unit of analysis, and encompassed the interviewee's answer to a question) independently before calibrating with the other coders. As part of calibration, the first four authors conducted a mini-workshop to discuss their understanding of the codes after coding each transcript. We defined coder agreement as any dialog segment where all 4 coders had at least one code for the segment in common (we merged synonymous terms into one single code). Since our mini workshops involved extensive discussion on the meaning and use of the codes in our codebook, the coders evolved a shared understanding of each code.

Following the first phase of coding, we coded an additional seven interview transcripts distributed across the three organizations (P3, P4, P9, P12, P13, P16, and P18). The

2. Full set of questions are in our research artifacts repository

TABLE 2
Progression from raw text to code to theme. One snippet of raw text is related to one or more codes, and one theme relates to many codes.

Raw Interview Text	Codes	Theme
"We have a metric that tracks it to monitor it. So what happens is on a deployment I would monitor it and it is actually part of my responsibility to monitor over the day and see how that how those numbers are relative to the previous day."	Metrics, Implicit, Deployment, Organizational, Manual, NFR-Perception, DevOpsPerception, Testing	*Put a Number on the NFR*

TABLE 3
Codebook Examples

Code Name	**Description**
Configuration-ManagementNFR	NFR: CONFIGURABILITY
NFROffloading	Relinquishing technical control / responsibility of said NFR to an external entity
NFRPerception	Individual's perception of NFRs
PerformanceNFR	Performance of the outcome (ie the output) usually measured by time
Metrics	Creating or monitoring of a quantitative value
Tooling	"Off the shelf" (Kubernetes, Docker, etc)

intention was to further develop codes and consistency in coding. In this stage, each interview was individually coded by two separate authors; inter-rater agreement levels varied from 64% to 93%, with an average agreement of 85%.

In the last phase of coding we divided the remaining interviews and assigned a coder for each interview. The number of codes created in the last 10 interviews accounted for only 4 additional codes, thus our codebook was saturated. To ensure that a coder did not miss any important codes during individual coding, we included a sanity check step where another coder would check the first coder's results. Table 3 shows a sample of six codes from our codebook, while the entire codebook (61 codes) can be found at in our research artifacts.

To answer RQ1 and RQ2 we developed themes based on thematic synthesis [19] of our coded data. We discussed similarities and differences between codes to group codes into clusters [44], whereby each cluster had a distinct higher-order theme. Each theme represented either a practice (RQ1) or a challenge (RQ2). Table 2 shows an example of relating a raw transcript quote to an eventual theme. Finally, to increase the credibility and validity of our research findings, we performed member checking [45] with the study participants to verify whether our findings resonate with the context of their organization. The member checking feedback was used to revise our findings.

4 Study Results

In this section we describe the themes (4 practices and 3 challenges) we derived from our analysis. We summarize these practices and challenges in Table 4.

All three software organizations in our study care deeply about NFRs in their software development. Alpha, in the data business, is most concerned about CONFIGURABILITY, SECURITY, and SCALABILITY. These three NFRs are vital to Alpha's business as Alpha processes and stores data from large numbers of users per day on a third-party hosted infrastructure. Beta, a leading e-commerce platform, is primarily concerned with USABILITY, PERFORMANCE, and STABILITY/RELIABILITY, as their applications handle millions of commercial transactions per day. Finally, Gamma, an online advertisement content provider, requires PERFORMANCE, REVENUE, and CONFIGURABILITY to ensure that Gamma's infrastructure can effectively deliver content to a wide audience.

Table 5 shows the overall ranking and individual ranking for each organization based on the frequency that the code representing the NFR appeared in our data (with at least 30 occurrences). The entire codebook can be found in our research artifacts. Gamma's number 2 ranking (REVENUE) is notably absent from the table due to less than 30 overall occurrences, despite the high number of occurrences at Gamma.

Throughout our paper (and in Table 5), we link our codes or particular findings to the evidence (interview text) as much as possible using quotes as well as spark-histograms. First introduced by Ying and Robillard [46], these histograms represent a compact and quick form of assessment of our findings: each histogram captures the 18 participants interviewed on the x-axis, while the y-axis shows the number of times the given code was mentioned by each participant, relative to the total number of mentions of that code (i.e. normalized). We also show the total number of mentions following the histogram.

The x-axis is ordered to match Table 1. For example, consider the histogram for our code CONFIGURABILITY ( , 103 mentions), which shows participants 3-6 mentioned this code more often than the other participants, and across all transcripts, this code occurred 103 times.

4.1 Practices For Handling Non-Functional Requirements in Continuous Software Engineering (RQ1)

Our first research question examines how NFRs are managed within each organization—the practices they use to handle NFRs in the context of agile teams following CSE.

TABLE 4
Practices and Challenges of Organizations Handling NFRs with CSE

Practices	Put a number on the NFR Let someone else manage the NFR Write your own tool to check the NFR Put the NFR in source control
Challenges	Not all NFRs are easy to automate Functional requirements get prioritized over NFRs Lack of shared understanding of an NFR

TABLE 5
Ranking of NFRs by frequency of mention. Sorted by overall frequency. Columns Alpha, Beta, & Gamma refer to rank within each organization. (T indicates a tie)

Rank	NFR	Histo.	Alpha	Beta	Gamma
1	Configurability		1	T12	3
2	Performance		T4	2	1
3	Security		2	T5	6
4	Scalability		3	T5	5
5	Usability		T14	1	T9
6	Reproducibility		T4	T5	4
7	Testability		T6	4	8
8	Stab./Reli.		10	3	T9
9	Availability		T6	T5	T12
10	Maintainability		8	T5	T17
11	Readability		9	T15	T14

We identified four practices from our analysis. These were: *Put a number on the NFR, Let someone else manage the NFR, Write your own tool to check the NFR*, and *Put the NFR in source control*. We discuss each practice in turn.

4.1.1 Put a Number on the NFR

The first practice for dealing with NFRs is establishing metrics to help validate and assess a particular NFR. The concept of NFR metrics was frequently discussed during our interviews (, 154 mentions). When an organization tracks pertinent metrics for NFRs, metric indicators collected by the organization's deployment pipeline provide valuable insight on NFRs.

For Gamma, PERFORMANCE is the most important NFR and is primarily tracked through a *"[caching ratio] that we have from [redacted] our caching provider goes [on a dashboard] because that's usually a pretty good indication that something has gone wrong. It also drastically affects* PERFORMANCE*"* (P14). At Alpha, a drop in response times below a specific metric will cause a decline in PERFORMANCE. As a result, they *"have a certain amount of monitoring set up [...] You're also defining which alarms are set. Therefore, if requests [drop] below [redacted] milliseconds and that's what you want to hold it to. Then that would be codified in the alarm"* (P5). Without setting a quantitative metric, Alpha may not receive warning that its software experienced a dramatic drop in PERFORMANCE.

Similarly, Beta provides a platform for many retailers and USABILITY is important for those retailers. As part of managing USABILITY, Beta tracks the number of customer actions required to complete a transaction through the use of USABILITY metrics: *"I can tell you that 90% of our customers have less than [redacted] items [...] They'll [say] we know that each [order] requires [redacted] page loads"* (P7). While this may only represent a subset of tasks for USABILITY, Beta views it as part of 'satisficing' USABILITY, in particular by bringing awareness to other teams. While at Gamma metrics are a key component in managing USABILITY, across cross-functional teams, including development and product management by *"showing how many users are using a specific feature, where the feedback benefits developers but our product team in their ability to make their decisions"* (P16).

A critical success factor putting a number on the NFR is the feedback loop (, 46 mentions) to enable the continuous monitoring of metrics. The feedback loop is an integral part of CSE [16] and is one of the earliest perceived benefits of adopting CSE. For Alpha, which deals with a lot of user data, effectively managing SECURITY awareness is important for its business, e.g. *"at least people who need to be aware of IAM changes are automatically notified"* (P3). At Beta, the feedback loop provides the ability to quickly identify bugs that crash the software: *"I think quick feedback is the core of DevOps so that we will see what broke"* (P6) and *"it reduces risk because things are integrated more frequently"* (P6). In particular, the feedback loop is most useful if it is a *quick* feedback loop and many organizations strive to reduce the time required for a feedback loop to complete, e.g. *"developers really want [the] feedback loop to be tight"* (P13).

4.1.2 Let Someone Else Manage the NFR

The most popular approach to managing NFRs we found was to let someone else do it by offloading it (, 155 mentions), where the 'someone else' is typically a large cloud-service provider. Offloading an NFR means that an organization allows another platform or tool (, 262 mentions) to realize, at least part of, the NFR on the organization's behalf.

An organization's ability to focus on core functionalities and behaviour of their software is heightened by offloading the brunt of the work to realize NFRs such as SCALABILITY, RELIABILITY, etc. However, offloading an NFR is not as simple as flipping a switch to realize an NFR. In many cases some form of configuration is required; furthermore, ensuring the system is ready for configuration, i.e., is configurable, requires a specialized skill set.

CONFIGURABILITY was the most referenced NFR (, 103 mentions). We found that the organizations we studied made frequent use of CONFIGURABILITY tooling such as Docker, Terraform, and Kubernetes. These tools help Beta maintain three separate software stacks through dependency management, which allows developers and testers to easily create or re-create an environment and application, e.g. *"it automatically does it in the Docker compose files. So we've automated a lot of dependency updates and stuff like that"* (P11). Furthermore, cloud providers are favoured by DevOps engineers as they have access to a cloud provider support team to assist with issues, *"cloud providers are awesome. I love being able to just file a support ticket"* (P12), as opposed to dealing with an issue on their own. Finally, CONFIGURABILITY tools also help manage REPRODUCIBILITY (, 52 mentions), e.g. *"the most important part is that the builds and the environment they run on and deploy to are defined in a repeatable way or state"* (P5).

SECURITY (, 68 mentions) was also offloaded to a third-party service. SECURITY and PRIVACY checks may be intrinsically supported by a third-party service, reliving an organization of the obligation of laboriously maintaining and provisioning these checks on its resources. SECURITY aspects can be offloaded, e.g. *"I believe [*SECURITY*]'s all codified in like the Docker and Kubernetes world"* (P6). Gamma utilizes a security key management system directly from one of the cloud providers.

Cloud providers are heavily relied upon to manage PERFORMANCE (, 88 mentions), AVAILABILITY (, 39 mentions), STABILITY/RELIABILITY (, 41 mentions), and SCALABILITY (, 56 mentions). For example, at

Alpha, *"[your web application is] immediately spread across however many* AVAILABILITY *zones and nodes as you want"* (P5).

Finally, NFR offloading provides the capability to ratchet NFRs [25], as all three organizations highlighted their ability to increase the amount of resources they consume from their cloud providers if they hit a particular NFR bottleneck. For example, Gamma and Alpha both noted that they can simply pay more to the cloud provider, e.g., *"We just put money in the machine and made it better"* (P5).

4.1.3 Write Your Own Tool to Check the NFR

As opposed to offloading an NFR to a third-party service or tool, organizations also wrote custom, in-house source code, custom tooling, scripts, or manifests (, 78 mentions). Beta and Gamma both codified some USABILITY parameters of their user facing front-end to manage a portion of USABILITY according to their individual definition and satisfaction, which may not be broadly applicable. This codification was done as part of their source code. They submit the source code, wait for the source to process through the CSE pipeline, and finally observe and verify the result, *"this is how we define* USABILITY *and make sure that it's there. If you want to change our* USABILITY *parameters or whatever we change it in source code and then we can test it and verify that it still meets our needs"* (P10).

For a resource-constrained organization a custom tool is usually a last resort, where the existing off-the-shelf solutions either do not exist or do not sufficiently meet particular requirements. A custom tool may be based on an augmented third-party tool that requires significant modification to meet the specified needs, e.g. *"so we used to have [name redacted] dashboards out there ... [but that] didn't give us any application specific information"* (P14).

Some custom tools were used to handle NFRs from an operations perspective to determine AVAILABILITY (, 39 mentions) or STABILITY/RELIABILITY (, 41 mentions) of the infrastructure. Other custom tools were developed to help automatically enforce or validate SECURITY (, 68 mentions) within a CSE pipeline: *"so there's a lambda [function] that runs when you make a bucket, it triggers and goes 'you didn't encrypt', it turns on encryption, tells you, 'you are an idiot'.* SECURITY *non-functional requirement!"* (P4).

4.1.4 Put the NFR in Source Control

At the organizations we studied, the workforce was constantly changing (typically growing), and the products were experiencing a rapid pace of change. The constant change requires that developers gain a clear understanding of the NFRs of the product so they know how their changes might impact these NFRs. Typically, an NFR is not effectively communicated via natural language documents: *"with respect to codifying something vs documenting it: it's not precise enough it's written in English. It's open to interpretation."* (P17)

Our results show developers captured NFRs directly in source control. 'Codification' refers to using code and related artifacts (such as version-controlled JSON configuration files) to capture NFR knowledge (, 92 mentions). For example, automated dashboards monitor health indicators, such as AVAILABILITY, and these explicit rules or triggers that represent metrics of an NFR are in source-control. Codification helps set an objective metric for developers who might not have had enough time to acquire the tacit knowledge about what constitutes an acceptable NFR threshold. P17 notes *"my understanding is that we should be able to see our tests in source control in terms of nicely capturing results in a way that I and the other developers can see and say some data is not captured yet"* (P17).

4.2 Challenges (RQ2)

While the organizations in our study have concrete practices for managing NFRs, they still face challenges (, 71 mentions). The most often described challenges were *difficulty using tools and tests with NFRs*, *difficulty prioritizing NFRs*, and *challenges with knowledge transfer*.

4.2.1 Not All NFRs are Easy to Automate

Some NFRs, such as USABILITY, are intrinsically difficult to verify through automated means. Unfortunately, adopting CSE approaches to development, which means committing and deploying at an extremely rapid rate, appear to make this an even greater challenge. For example, Beta relies on some manual USABILITY acceptance verification for any customer-facing software deployment. As a result the people tasked with manual verificationour findings suggest of USABILITY become a bottleneck in CSE: *"[User acceptance testing is] easier to chunk into one deployment rather than 12 a day."* (P7).

In addition to USABILITY, it may be difficult to write automated tests for other NFRs: *"There's a lot of variability and it's hard to write really good thresholds of what is working. What is not working"* (P14). Although some NFRs may not require much work to automate, based on the interviewees' sentiment, producing the "right" automated tests may require more than one test creation iteration.

Furthermore, purely increasing the number of tests does not equate to higher quality tests: *"I think that testing itself [is a] quality metric. So we're looking at coverage as a possible metric but we're trying to determine a more accurate form because we don't believe that [increasing] code coverage [will] provide us the value [...] if you test all cases that you have 100% coverage, it's not really scalable"* (P16).

Under normal circumstances, it can be difficult to predict a sudden spike in user activity. If Gamma suddenly experiences overwhelming levels of traffic, prior tests for PERFORMANCE and SCALABILITY may not suffice: *"I feel like [tests] always [had] a bias toward the happy cases [...] When something does go wrong, it's something horribly out of left field [...] How do you prepare for those? How do you think about what left field is?"* (P18). Determining the parameters and conditions that would effectively verify the entire problem space of an NFR is difficult.

4.2.2 Functional Requirements Get Prioritized Over NFRs

Since our collaborating organizations are still growing rapidly, employees balance many other responsibilities, among other potential limits to resources. In resource-constrained environments, NFRs are an easy target to bump from the sprint plan or milestone due to lack of clarity around how to verify or define an NFR. At Alpha, one

aspect of their system is the need for CPU power to process large volumes of data. However, if not enough developers are available to maintain the system, EFFICIENCY and PERFORMANCE may degrade over time: *"nobody's looked at this last six months maybe someone should check it out [...] It's a resource management issue and a lot of times you have too many things for too few people."* (P4). An organization may be obliged to make NFR trade-offs with the hope that immediate, short-term success will lead to the ability to remedy the trade-offs, i.e., potential technical debt, in the future. In our study, 14 out of 18 interviewees ( , 41 mentions) acknowledged the existence or previous existence of trade-offs: *"the sort of startup code today some of it you could consider clever but you do too many clever things then rack up a lot of technical debt"* (P5).

An organization, even an early-stage organization, needs to be aware of these NFR trade-offs so that it can ensure that an important NFR such as SCALABILITY is improved when the NFR reaches a low point: *"Some of the core pieces of the system again get more love or more time to knock that [technical debt] number down or we just pay more close attention"* (P5).

4.2.3 Lack of Shared Understanding of an NFR

A shared understanding of an NFR implies that everyone involved with the NFR is in agreement with the meaning of that NFR, and its various components. This shared understanding relies on knowledge transfer from the people—such as the product manager or CEO—who created the NFR, to those whose work might affect it. Shared understanding is a challenge for our subject organizations. They faced problems with inconsistency in what gets explicitly stored in source control; with tacit knowledge and a low (bad) circus factor [47]; and problems with role siloing.

Our study found organizations relied inconsistently on documentation for knowledge transfer of NFRs ( , 78 mentions). Explicit NFR knowledge transfer ( , 55 mentions) occurs when a developer relies on a formal metric or artifact, such as documentation, to frame their understanding of whether an NFR is being met. For instance, in reference to having their infrastructure run by Terraform scripts to deploy using Kubernetes, one of our respondents mentioned: *"a big incentive for me is the idea that I don't become a linchpin and at the same time a bottleneck being the one person specialized in this"* (P3).

Our subject organizations do not consistently invest effort or resources in documenting NFRs. While some NFRs are being actively monitored (see section 4.1.1) or are documented as code within a source control system (see section 4.1.4), others are not. For example, in reference to the fact that the PERFORMANCE of a feature requires processing to finish within 2.5 hours: *"No, I don't think I have that specifically labeled. I don't think I outlined any specific requirement like that"* (P2).

Implicit knowledge transfer of NFRs ( , 29 mentions) occurs when someone on the development team attains a personal understanding of an NFR without relying on an established metric or artifact. For example: *"at the moment it is tacit knowledge and unfortunately when a new developer comes in and starts working on stuff [they struggle]"* (P10). The 'circus factor'[3] [47] captures a major problem with implicit knowledge: *"I try not to get hit by a bus. So does [redacted]. Certain parts of the system are maintainable because certain people know how they work versus it being [explicitly] documented"* (P5).

Implicit knowledge is often obtained by one or two people who have the overarching view (typically early employees or founders). *"I know a lot of the team leads have a ton of non-functional requirements in their head and how things should work and they're kind of the ones gating what goes out based on those undocumented non-functional requirements. If we were to document those we could get those ideas into the heads of the people actually writing the code, and better the development experience"* (P13). Without transferring the knowledge of NFRs to front line developers, awareness of the importance of particular NFRs is lost.

While explicit knowledge may be in source control (see Section 4.1.4), role siloing makes understanding the artifacts difficult ( , 51 mentions). For example, capturing NFRs in the deployment scripting language Terraform is likely highly useful for team members working closely with deployment and DevOps roles, e.g. *"For anything that I do, [infrastructure as code] ends up in Terraform as a form of documentation"* (P3). However, the value of documentation provided solely by code can vary depending on developer context. Developers less familiar with Terraform may have different interpretations of what the script is doing, if they can understand the Terraform language in the first place.

5 DISCUSSION

Our empirical study sought to unveil the state of practice in managing NFRs in organizations that use CSE. In particular, we studied three organizations, each developing software with several vital NFRs and exhibiting mature continuous software engineering. Our findings suggest that the studied organizations manage NFRs in CSE through four main practices. We believe these practices are best-practices for managing NFRs, as our respondents, overall[4], are very satisfied with how their respective organization manages NFRs. While these practices may not be specific to CSE, we believe a special relationship exists for each that is unique in a CSE context.

While the use of metrics is well-established in industry, CSE enables an organization to better automate and deploy metrics in a rapid feedback loop. Furthermore, while NFRs are typically more difficult to automate, CSE brings a heightened focus, attention, and importance to automating important NFRs. While the lack of shared understanding of NFRs has been commonplace, we found evidence to suggest the CSE has led to a decrease in shared understanding [48]. Finally, while FRs may be often prioritized over NFRs, the fast paced environment of CSE exacerbates the deprioritization of NFRs.

Our results allow us to reflect on the somewhat surprising opportunities that CSE practices offer to managing NFRs, as well as associated challenges and trade-offs when

3. We avoid the ugly implications of 'bus factor' in favor of circus factor: the number of people who have to run away to join the circus to hurt the project.

4. 11/12 respondents believe NFRs are well managed

managing NFRs in CSE. We also discuss the importance of CONFIGURABILITY as an NFR in organizations practicing CSE. Our findings represent valuable empirical insights that add to the nascent empirical evidence on utilizing CSE to manage NFRs [17]. Finally, we propose research directions for the treatment of NFRs in CSE.

5.1 Non-Functional Requirements in Continuous Software Engineering: A Silver Lining

NFRs often do not get the appropriate attention they deserve. NFRs are cross-cutting in nature as they impact many aspects of the system and may be difficult to decompose into fragments that can be realized in a short, rapid CSE iteration [49], which further complicates the ability of an organization to manage an NFR. However, our evidence suggests that it might be easier, for organizations that shift to CSE, to manage through one of the four practices we identified ("Put a Number On the NFR", "Let Someone Else Manage the NFR", "Write Your Own Tool to Check the NFR", and "Put the NFR in Source Control").

Typically managing an NFR encompasses a number of steps, including elicitation, analysis, negotiation, implementation, verification, and validation. However, these practices at our organizations suggest that an NFR may be "realized" *without direct* implementation, i.e. actions have been taken to satisfy that the conditions of the NFR have been met, although it may not necessarily be implemented or verified by the organization. Hence we use "realization" as a broader term, as opposed to the traditional implementation. The "realization" of an NFR is composed of a number of subtasks and in this paper we use realization to indicate when an organization has reached a satisfactory level of an NFR to have "realized" it, whether or not it is completely 'satisficed' or not. However, during the process of "realizing" an NFR, it is vital to "put a number on the NFR", as a "realized" NFR may be affected, perhaps negatively, unbeknownst to developers (e.g. the implementation of a new feature causes PERFORMANCE to crater).

While our study did not directly observe the elicitation of NFRs at the three organizations, the practices we identified were concrete actions these organizations took to support NFR realization and verification. While the recent comprehensive SLR [17] confirmed the ability to *verify* NFRs by leveraging CI, this is just one aspect of NFR management. NFR verification may confirm *if* an NFR has been realized; however, realizing an NFR doesn't necessarily mean the NFR is verified.

For example an organization may realize, potentially only a part of, RESILIENCY by offloading it to Amazon Web Services and potentially verify RESILIENCY with some form of chaos engineering [50]; however, note that realization and verification do not necessarily go hand-in-hand. As part of NFR management, we found an organization was able to realize an NFR, for example AVAILABILITY, RESILIENCY, or SCALABILITY by offloading to a third-party, ultimately resulting in very little overhead (aside from cost) to the organization.

While the SLR by Yu et al. [17] is the closest work to ours, they found leveraging CI is underutilized to verify NFRs and that the ratio of industrial to theoretical studies is low—thus highlighting the importance of our study bringing substantial empirical evidence to support that CSE is an enabler in, not only testing but, realizing NFRs.

Furthermore, through our study of the practices and challenges at these organizations we uncovered 30 NFRs that they found relevant (which are clearly not complete for *all* organizations). Seven NFRs are in common with the findings from Yu et al. [17] (LATENCY and PRODUCTIVITY being the exceptions). Notably absent from their list are 3 of our top-5 NFRs, namely CONFIGURABILITY, SECURITY, and USABILITY.

Although Yu et al. specifically mention USABILITY as hard to verify, our study provides evidence to suggest that some organizations are satisfied with their level of realizing USABILITY; of course this distinction is relative and may not apply to other organizations in such a black and white manner. In particular, we previously discussed how Beta was able to leverage CSE to realize USABILITY in real-time (see Section 4.1.3). In addition, Gamma is able to track USABILITY metrics through their CSE practice, including user events, such as button clicks, page views, and navigation traces, and runs large scale A/B experiments [51]. While, this distinction certainly merits further investigation to *exactly* how this organization satisfactorily realizes USABILITY and how this realization can be applied to other domains and organizations, the exact details are outside the scope of this paper. By leveraging metrics, the feedback loop, and continuous monitoring, Beta and Gamma have near-constant realization of USABILITY—an otherwise difficult to realize NFR.

Metrics, of any kind, are the starting point that allow an organization to set goals, track progress, and monitor the state of the system in a reliable fashion [52]. However, metrics are not without problems, as assigning a desired threshold to an NFR is not trivial. Fixed or static thresholds may be problematic for complex NFRs, requiring alternative solutions such as desired, minimum, dynamic thresholds [53], or even the use of artificial intelligence to adapt the thresholds.

We believe that continuous, rapid iterations using metrics, the feedback loop, and continuous monitoring brings an increase to transparency and traceability of NFRs. Transparency and traceability are afforded by allowing anyone in the organization to easily track changes to a particular NFR metric, back to a localized source commit in the code [54]. Traceability in CSE has been previously studied in the Eiffel approach [31]; however, the Eiffel approach is aimed at improving the CSE pipeline, not necessarily the resulting software. The authors [31] note that further work includes extending Eiffel to consider development activity, which would ideally include NFR activities as well.

The ability to realize and continuously monitor, track, and audit NFRs in real-time throughout the entire life cycle of a software project is immensely powerful [31]. Alternatively, an organization may hire consultants to assess satisfaction of a particular NFR, such as SECURITY; however, this is often a one-time assessment and does not help with *continuous*, ongoing satisfaction of the NFR in question, which is usually the key from an operational point of view.

While assigning metrics to NFRs is not a new idea and has always been a recommended practice to ensure proper verification of NFRs [55], consideration of the metric often

only happens during the initial design and architecture phases. Once the NFR has been defined, measured (and perhaps satisfied), and the organization is deeply entrenched in actual coding, the NFR may no longer be tracked [3]. The key novelty with CSE is that it facilitates realizing NFRs and the constant and continuous ability to monitor and satisfy NFRs through the quick feedback loop. An organization is able to look at their CSE pipeline and determine the gap between NFR objectives, and actual level of PERFORMANCE, USABILITY, or CONFIGURABILITY (among others).

Research Implication 1

Our research has highlighted how CSE has enabled organizations to realize NFRs (to varying degrees) through the four practices highlighted in our research. While realizing NFRs in CSE is a promising trend, future empirical studies should seek to investigate the impact of continuously monitoring NFR satisfaction on software design.

5.2 Trade-Offs in Realizing Non-Functional Requirements in Continuous Software Engineering

While we have shown that CSE further enables an organization to realize NFRs, from our findings we uncovered three notable trade-offs. For each trade off, we discuss our findings in relation to relevant existing literature and highlight areas worthy of further research.

5.2.1 Offloading NFRs to Third-Party Providers Results in Losing Control Over an Offloaded NFR

The emergence of cloud providers, such as AWS, Google Cloud, and Microsoft Azure, offers significant advantages to software development organizations. First, it has encouraged and facilitated small organizations to realize NFRs, perhaps only partially through sub-tasks, that would otherwise not be within their reach, such as SCALABILITY. Second, offloading sub-tasks of NFRs, such as SCALABILITY and PERFORMANCE, enables an organization to devote additional resources to enhancing the core product [56] and is key to a small organization's business success. Often, finding money to pay for NFR offloading is easier than finding staff or time, especially for small, resource constrained organizations. Furthermore, there is some notion that some NFRs may be realized and guaranteed through certified quality of service guarantees [57], allowing an organization to focus on the core of their business. Third, the utilization of cloud platforms [58], or even simulators [59], allow an organization to easily build otherwise costly environments solely for the purpose of verifying RELIABILITY, AVAILABILITY, PERFORMANCE, and SCALABILITY.

At the same time we must recognize that offloading an NFR may not imply the NFR is realized across all aspects of the organization. This is even more important with the prevalence of distributed or micro-service architectures, as offloading an NFR for one particular component does not satisfy that NFR for the entire system. Given the cross-cutting nature of NFRs, one must recognize the limitation of offloading an NFR and carefully plan to ensure that offloading will actually achieve a desired result. Organizations must be cognizant of the limitations of offloading an NFR.

We identified two costs associated with offloading an NFR, 1) the loss of control of the NFR and 2) the potential for vendor lock-in. First, the offloading organization will be at the mercy of the organization taking over that NFR [60]. If an NFR is realized by decomposing that NFR into a series of sub-tasks, then an organization might lose control of those sub-tasks, or the assigned priority of those sub-tasks. In particular, if an organization has offloaded a portion of and NFR, such as AVAILABILITY, to a cloud provider and that cloud provider experiences an issue, such as an outage, the organization will also experience an outage and hence the AVAILABILITY of the organization's product is now entirely out of their control [61].

Second, with offloading there is also the risk of vendor lock-in, which occurs when a customer is overly dependent on a vendor, such as a cloud provider, and is unable to switch to another vendor without substantial re-work. Vendor lock-in has long been a problem in the software industry [62]; interestingly, we did not hear about vendor lock-in from any of our interviewees. However, neither cloud provider tools nor standards are widely adopted (see Section 5.3) so the potential for vendor lock-in remains and is an area of active research [63], [64], [65].

5.2.2 CSE Hurts Shared Understanding of the NFR

While shared understanding is a critical success factor in producing high quality software designed to meet stakeholders' needs [66], our study highlighted a lack of shared understanding of key NFRs in each of the three organizations, despite their ability to leverage CSE to realize the NFRs. For example, our organizations described how role siloing between DevOps and developers creates a lack of shared understanding of NFRs. However, this phenomenon lacks substantial empirical research [67], as the practice of creating and maintaining a shared understanding in agile is not well established [68].

In a follow-up investigation on this trade-off, we sought to further understand and quantitatively validate details of this lack of shared understanding from an analysis of the project management repositories at these organizations [48].

Our analysis identified that while there is an acceptable and unavoidable amount of lack of shared understanding, which captures unknown unknowns [69] and represents desirable learning and feedback [23], 78% of the lack of shared understanding was deemed *avoidable* by the three organizations. These results bring additional evidence that an organization realizing NFRs in CSE may do so at the cost of a lower shared understanding of those NFRs.

5.2.3 Fast Pace of CSE Deprioritizes NFRs

Agile methodologies have been shown to risk the overemphasis of FRs at the expense of NFRs [70]. Our findings corroborate previous research [35], [71] indicating that the fast pace of CSE increases the risk of deprioritizing NFRs. Our results also suggest that neither frameworks nor models produced from research are adopted in practice to assist prioritizing NFRs.

While the developers we interviewed indicated that NFRs are important (to developers), the perceived importance of NFRs differs for product managers, among others. As such, NFRs were largely left to the developers to

self-manage in an ad-hoc manner. Despite the numerous frameworks and models developed through research [72], [73], [74], [75], there exists a gap between industry and practice on whether they can actually be used to solve this issue of NFR prioritization. This is a significant empirical finding adding to the scarce evidence on how (the lack of) NFR prioritization is handled in industrial versus research settings.

> **Research Implication 2**
>
> Realizing an NFR through CSE comes at the cost of substantial trade-offs, such as lower priority and a lack of shared understanding of NFRs, or for organizations leveraging offloading to third-parties, loss of control of the NFR. Research is needed to develop and empirically evaluate mitigation strategies or techniques to reduce the risk of and to overcome these trade-offs.

5.3 The Importance of CONFIGURABILITY as a Non-Functional Requirement

The importance of CONFIGURABILITY as an NFR grows as an organization relies more heavily on CSE practices. CONFIGURABILITY is an attribute of the software system that refers to how easily an organization can configure its software infrastructure and environments [55], including "Infrastructure as Code" (IaC). However, CONFIGURABILITY is *more* than just a system quality, as it also encompasses overarching *process quality*. Our data show the importance of the non-functional quality of the system's configurability—the source code, build scripts, infrastructure and deployment configuration, and associated hardware. Like maintainability, configurability refers to an internal quality that supports the goal of rapid deployment and re-configuration. To realize this NFR, one might use tactics such as rollbacks, keeping production and development environments in sync, or applying infrastructure as code tools such as Puppet.

Comprehensive CONFIGURABILITY has long been considered to be an enabler of the many perceived benefits of CSE by Humble et al. [76]. However, CONFIGURABILITY is largely underrepresented. The concept of CONFIGURABILITY has further grown to encompass the configuration of build, staging, and deployment infrastructure, ideally with little to no human intervention [77]. Our study brings clear evidence that CONFIGURABILITY should be considered an extremely important and high priority NFR in CSE.

As software systems increasingly exist as a service running in the cloud, application code is no longer the only important source code. Infrastructure configuration and code are as vital to software business goals as application code.

At Alpha, customers are in part paying for Alpha to host reports and data for them. Thus, their infrastructure configuration and code must also exhibit NFRs such as RELIABILITY and AVAILABILITY. In contrast, these NFRs are entirely the customer's responsibility for an on-premise offering. An organization must now invest in CONFIGURABILITY in parallel with other NFRs and features.

As the technology director at Gamma commented during our member checking phase, *"[*CONFIGURABILITY*], and associated IaC and automation is the enabler that allows organizations like ours to essentially offload other NFRs such as* AVAILABILITY, SCALABILITY, *and* SECURITY *to cloud providers [...] without [*CONFIGURABILITY*] other NFRs would suffer, such as* RELIABILITY, MAINTAINABILITY, REPEATABILITY, *and even* AVAILABILITY *due to more human error during deployments."*

The increased reliance on CONFIGURABILITY results in a trade-off: the organization needs to now spend significant additional resources on configuration, developer training, and avoiding potential vulnerabilities associated with CONFIGURABILITY, including the lack of shared understanding.

First, CONFIGURABILITY now has its own set of distinct code-smells [78]. There are early efforts to mitigate these code-smells, such as Rahman et al. [79] to identify code-smells of CONFIGURABILITY code in open source software. Second, while CONFIGURABILITY may benefit from standard coding practices it is not yet done in practice [63]. Conversely, CONFIGURABILITY is actually associated with a wide variety of disparate languages and tools. Most organizations use three or more different tools and no single tool is used by the majority of organizations [63]. Existing standards, such as Topology and Orchestation Specification for Cloud Applications (TOSCA) and Open Cloud Computing Interface have been proposed; although the adoption amongst DevOps engineers is low (18%) [63]. Third, the standards themselves do not contain a complete set of NFRs, as they require extensions to include SECURITY [80], [81] and PRIVACY [82], among others.

Our study highlights the need to fill in the gap between research and industry efforts on supporting CONFIGURABILITY. As more and more configuration is stored as code (IaC), the same problems we see in traditional NFRs is likely to surface in CONFIGURABILITY, including the aforementioned trade-offs (e.g. loss of control and shared understanding).

> **Research Implication 3**
>
> CONFIGURABILITY is emerging as a vital NFR and an integral part of developing software; however, it is yet understudied as an NFR. In-depth research is required to develop and evaluate techniques to manage CONFIGURABILITY, including elicitation, analysis, validation, and verification. Further empirical studies are needed to explore and propose solutions to the associated trade-offs and challenges with CONFIGURABILITY.

5.4 Implications for Practitioners

In addition to researchers, our study has wide reaching implications for practitioners. Our observed practices and challenges for handling NFRs in CSE demonstrate that organizations are both successfully treating NFRs and encountering difficulties. For practitioners, there are three noteworthy implications. First, organizations must be aware of the ability to realize NFRs using the four practices that leverage CSE, but also be mindful of the associated challenges. Second, while offloading NFRs to a third party provider has many potential benefits, practitioners should be aware of the potential consequences of offloading. Furthermore, practitioners should monitor any offloaded NFR to ensure the NFR is

treated as expected. Finally, practitioners need to recognize the importance of CONFIGURABILITY and dedicate time to educate, elicit, analyze, and verify CONFIGURABILITY.

6 THREATS TO VALIDITY

Threats to validity in qualitative research typically concern the reliability of the results. We use the total quality framework of Roller [83] to discuss these potential threats and our mitigation strategies. The framework is a way to assess the quality of qualitative research using four categories: credibility, analyzability, transparency, and usefulness.

Concerning the *credibility* of our study and data gathering methods, our study investigated the state-of-the-practice at three industrial organizations performing some form of CSE. Our selection of these three organizations might suffer from sampling bias, as we chose organizations willing to participate from a larger pool of local companies. However, our preparatory study phase ensured that they practiced CSE inline with industry and literature best practices to the best of our knowledge.

Our interviewees were representative of their organizations with respect to role, gender, and experience. The unbalanced distribution of roles (12 developers; 6 managers) and genders (13 males; 5 females) is representative of our organization's demographics, and unfortunately there were no other managers or females to interview. Our analysis did not reveal differences due to gender, role, or experience; this represents a worthwhile direction for future study.

To mitigate the threat of construct validity, we began each interview by examining the respondent's knowledge of NFRs. We then explained the NFR concept with examples so that each respondent had a similar level of understanding about NFRs. All our participants were proactive and valuable in offering details commensurate to their role and experience with their organization's practices.

As far as the credibility associated with data analysis, the primary threat is in the coding approach. We described our process in Section 3. We followed best practices for thematic coding and used the inter-rater agreement process frequently to align coding schemes. Due to the number of coders we allowed multiple codes to be applied to a unit, thus limiting our ability to apply an inter-rater agreement that would resolve chance agreement. Finally, we may be susceptible to researcher-participant interactions, since these were in-person interviews.

As for *analyzability*, we used computer-aided transcription, but we did check each transcript against the audio where the transcription was unclear. We utilized the open coding approach to remove the potential bias from coders. We also performed peer debriefings and analyzed deviant codes to verify our analysis and ensure our results were consistent and neutral.

With respect to *transparency*, we used histograms to enhance our thick descriptions of the responses. For reliability, we also conducted a member checking exercise to validate our findings with our subjects (12 of 18 interviewees responded). We elicited ordinal feedback (Strongly Disagree-Strongly Agree) on each of our practices and challenges. For all 4 practices and 3 challenges, the 12 respondents had a median score of "Agree". One challenge that we had originally included had a median score of Neutral, with 5/12 voting "Disagree" or "Strongly Disagree". As a result, we dropped this particular challenge pending further investigation. We were able to integrate additional insight (from one Director of Technology at Gamma) into our discussion of findings. We also make our codebook and analysis scripts available for replication in our research artifacts repository but due to NDA, we cannot share raw transcripts.

The *usefulness* of our study is geared towards bridging the gap between research and practice of handling NFRs for CSE organizations. In particular, we raise awareness of areas for researchers to focus on with respect to the current and emerging trends that can enable organizations to realize NFRs. We recognize that NFRs cannot and should not be grouped together, as the differences between individual NFRs can be as great, if not greater, than the difference between a FR and NFR; we believe that further in-depth studies should be focused on individual NFRs. While the usefulness to practitioners is to help bring focus to *how* an NFR can be realized and the associated pitfalls to each.

7 CONCLUSION

The effective management of NFRs is key to successful, high-quality software projects. NFRs themselves are well-known to be difficult to express, let alone manage, in part due to their cross-cutting nature. Since NFRs in the context of CSE have not been sufficiently explored in literature, we conducted a qualitative study to gather empirical evidence on how CSE organizations handle NFRs. Contrary to previous research, our investigation brings insights from three organizations that do manage NFRs using a variety of practices, yet continue to face important challenges and make trade-offs.

By discussing the practices and challenges from our findings, we also formulated research implications both for research and practitioners. While NFRs are difficult, ambiguous, and tough to verify in normal circumstances, we believe following the four practices will allow an organization to realize NFRs in CSE. In particular, our empirical evidence indicates that a key to rein in NFRs is to leverage CSE practices, such as the quick feedback loop or the capability to offload NFRs to third-parties. However, the peril of realizing an NFR by leveraging CSE is that an organization may lose control of an offloaded NFR, leaving them at the mercy of the third-party, or incur a decrease in the shared understanding of an NFR.

The practices and research implications we presented in this paper represent useful avenues for future research, in the form of hypotheses or methods to be validated through empirical studies in a broader set of, potentially larger, CSE organizations.

ACKNOWLEDGMENTS

We thank our three partner organizations for their time and collaboration. Our research was supported by Canadian grant NSERC-CRD 535876.

REFERENCES

[1] M. Glinz, "On non-functional requirements," in *International Conference on Requirements Engineering*, Oct 2007, pp. 21–26.

[2] L. Chen, M. A. Babar, and B. Nuseibeh, "Characterizing architecturally significant requirements," *IEEE software*, vol. 30, no. 2, pp. 38–45, 2012.

[3] A. Caracciolo, M. F. Lungu, and O. Nierstrasz, "How do software architects specify and validate quality requirements?" in *European Conference on Software Architecture*. Springer, 2014, pp. 374–389.

[4] E. Targett. (2018) Amazon outage: Estimated $99 million lost. [Online]. Available: https://web.archive.org/web/20190823180248/https://www.cbronline.com/news/amazon-outage-lost-sales

[5] D. Ocean. (2018) Currents: A quarterly report on developer trends in the cloud. Accessed: April 29, 2020. [Online]. Available: https://assets.digitalocean.com/currents-report/DigitalOcean-Currents-Q1-2018.pdf

[6] M. Fowler. (2006) Continuous integration. [Online]. Available: https://martinfowler.com/articles/continuousIntegration.html

[7] B. Fitzgerald and K.-J. Stol, "Continuous software engineering and beyond: Trends and challenges," in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, ser. RCoSE 2014. New York, NY, USA: ACM, 2014, pp. 1–9.

[8] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2016, pp. 426–437.

[9] W. Behutiye, P. Karhapää, L. López, X. Burgués, S. Martínez-Fernández, A. M. Vollmer, P. Rodríguez, X. Franch, and M. Oivo, "Management of quality requirements in agile and rapid software development: a systematic mapping study," *Information and software technology*, vol. 123, p. 106225, 2020.

[10] J. Eckhardt, A. Vogelsang, and D. M. Fernández, "Are "non-functional" requirements really non-functional?: An investigation of non-functional requirements in practice," in *International Conference on Software Engineering*. New York, NY, USA: ACM, 2016, pp. 832–842.

[11] B. Nuseibeh and S. Easterbrook, "Requirements engineering: A roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 35–46.

[12] W. Alsaqaf, M. Daneva, and R. Wieringa, "Quality requirements challenges in the context of large-scale distributed agile: An empirical study," *Information and Software Technology*, vol. 110, pp. 39–55, Jun. 2019.

[13] D. I. Mattos, J. Bosch, H. H. Olsson, A. Maryam Korshani, and J. Lantz, "Automotive a/b testing: Challenges and lessons learned from practice," in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2020, pp. 101–109.

[14] F. Chen, "From architecture to requirements: Relating requirements and architecture for better requirements engineering," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE, 2014, pp. 451–455.

[15] S. Wagner, "A literature survey of the quality economics of defect-detection techniques," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, ser. ISESE '06. Rio de Janeiro, Brazil: Association for Computing Machinery, Sep. 2006, pp. 194–203.

[16] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017.

[17] L. Yu, E. Alégroth, P. Chatzipetrou, and T. Gorschek, "Utilising ci environment for efficient and effective testing of nfrs," *Information and Software Technology*, vol. 117, p. 106199, 2020.

[18] R. K. Yin, *Case Study Research: Design and Methods*, 3rd ed. Thousand Oaks, Calif: SAGE Publications, Inc, Dec. 2002.

[19] D. S. Cruzes and T. Dyba, "Recommended Steps for Thematic Synthesis in Software Engineering," in *2011 International Symposium on Empirical Software Engineering and Measurement*, Sep. 2011, pp. 275–284.

[20] J. Aranda, S. Easterbrook, and G. Wilson, "Requirements in the wild: How small companies do it," in *International Conference on Requirements Engineering*. IEEE, 2007, pp. 39–48.

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

[22] N. Ernst and G. C. Murphy, "Case Studies in Just-In-Time Requirements Analysis," in *Empirical Requirements Engineering Workshop at RE*, Chicago, Sep. 2012, pp. 1–8.

[23] E. Ries, *The lean startup: how today's entrepreneurs use continuous innovation to create radically successful businesses*, 1st ed. New York: Crown Business, 2011.

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

[25] S. Bellomo, N. Ernst, R. L. Nord, and I. Ozkaya, "Evolutionary improvements of cross-cutting concerns: Performance in practice," in *2014 IEEE International Conference on Software Maintenance and Evolution*, Sep. 2014, pp. 545–548.

[26] A. Borg, A. Yong, P. Carlshamre, and K. Sandahl, "The bad conscience of requirements engineering: An investigation in real-world treatment of non-functional requirements," in *Third Conference on Software Engineering Research and Practice in Sweden (SERPS'03)*, 01 2003.

[27] D. M. Fernandez, "Supporting requirements-engineering research that industry needs: The NaPiRE initiative," *IEEE Software*, vol. 35, no. 1, pp. 112–116, Jan. 2018.

[28] S. Wagner, D. M. Fernández, M. Felderer, and M. Kalinowski, "Requirements engineering practice and problems in agile projects: Results from an international survey," in *Iberoamerican Congress of Software Engineering (CibSE)*, 2017.

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

[30] J. Bosch, "Continuous software engineering: An introduction," in *Continuous software engineering*. Springer, 2014, pp. 3–13.

[31] D. Ståhl, K. Hallén, and J. Bosch, "Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the eiffel framework," *Empirical Software Engineering*, vol. 22, no. 3, pp. 967–995, Oct. 2016.

[32] K. Petersen and C. Wohlin, "The effect of moving from a plan-driven to an incremental software development approach with agile practices," *Empirical Software Engineering*, vol. 15, pp. 654–693, 2010.

[33] A. Nilsson, J. Bosch, and C. Berger, "Visualizing Testing Activities to Support Continuous Integration: A Multiple Case Study," in *Agile Processes in Software Engineering and Extreme Programming*, 2014, pp. 171–186.

[34] M. Leppänen, S. Mäkinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M. V. Mäntylä, and T. Männistö, "The highways and country roads to continuous deployment," *IEEE Software*, vol. 32, no. 2, pp. 64–72, Mar. 2015.

[35] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, "Continuous deployment at Facebook and OANDA," in *Proceedings of the 38th International Conference on Software Engineering Companion - ICSE '16*. Austin, Texas: ACM Press, 2016, pp. 21–30.

[36] M. R. Barbacci, R. Ellison, A. J. Lattanze, J. A. Stafford, and C. B. Weinstock, "Quality attribute workshops (qaws)," CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, Tech. Rep., 2003.

[37] E. S. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Proceedings of ISRE'97: 3rd IEEE International Symposium on Requirements Engineering*. IEEE, 1997, pp. 226–235.

[38] H. A. Simon, "Rational choice and the structure of the environment." *Psychological review*, vol. 63, no. 2, p. 129, 1956.

[39] C. Pahl, P. Jamshidi, and O. Zimmermann, "Architectural principles for cloud software," *ACM Transactions on Internet Technology (TOIT)*, vol. 18, no. 2, pp. 1–23, 2018.

[40] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on software engineering*, vol. 25, no. 4, pp. 557–572, 1999.

[41] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting Empirical Methods for Software Engineering Research," in *Guide to Advanced Empirical Software Engineering*. Springer, 2008, pp. 285–311.

[42] C. Potts, "Software-engineering research revisited," *IEEE Softw.*, vol. 10, no. 5, pp. 19–28, Sep. 1993.

[43] J. M. Corbin and A. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative sociology*, vol. 13, no. 1, pp. 3–21, 1990.

[44] R. E. Boyatzis, *Transforming qualitative information: Thematic analysis and code development*. sage, 1998.

[45] M. B. Miles, M. Huberman, and J. Saldana, *Qualitative data analysis: A methods sourcebook*. SAGE Publications, Incorporated, 2013.

[46] A. T. T. Ying and M. P. Robillard, "Selection and presentation practices for code example summarization," in *ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014, 2014, pp. 460–471.

[47] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Assessing the bus factor of git repositories," in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, Mar. 2015.

[48] C. Werner, Z. S. Li, N. Ernst, and D. Damian, "The lack of shared understanding of non-functional requirements in continuous software engineering: Accidental or essential?" in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, 2020, pp. 90–101.

[49] S. Bellomo, N. A. Ernst, R. L. Nord, and I. Ozkaya, "Evolutionary Improvements of Cross-Cutting Concerns: Performance in Practice," *2014 IEEE International Conference on Software Maintenance and Evolution*, pp. 545–548, 2014.

[50] A. Basiri, N. Behnam, R. De Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos engineering," *IEEE Software*, vol. 33, no. 3, pp. 35–41, 2016.

[51] S. Gupta, L. Ulanova, S. Bhardwaj, P. Dmitriev, P. Raff, and A. Fabijan, "The anatomy of a large-scale experimentation platform," in *International Conference on Software Architecture (ICSA)*, Apr. 2018.

[52] S. Neely and S. Stolt, "Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy)," in *2013 Agile Conference*, Aug. 2013, pp. 121–128.

[53] K.-T. Rehmann, C. Seo, D. Hwang, B. T. Truong, A. Boehm, and D. H. Lee, "Performance monitoring in sap hana's continuous integration process," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 4, pp. 43–52, 2016.

[54] M. Rath, J. Rendall, J. L. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild: automatically augmenting incomplete trace links," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 834–845.

[55] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice* , 3rd ed., ser. SEI Series in Software Engineering. Addison-Wesley Professional, 2012.

[56] M. Anderson, "Performance modelling of reactive web applications using trace data from automated testing," Ph.D. dissertation, University of Victoria, 2019.

[57] M. Anisetti, C. A. Ardagna, E. Damiani, F. Gaudenzi, and G. Jeon, "Cost-effective deployment of certified cloud composite services," *Journal of Parallel and Distributed Computing*, vol. 135, pp. 203–218, 2020.

[58] R. Nouacer, M. Djemal, S. Niar, G. Mouchard, N. Rapin, J.-P. Gallois, P. Fiani, F. Chastrette, A. Lapitre, T. Adriano *et al.*, "Equitas: A tool-chain for functional safety and reliability improvement in automotive systems," *Microprocessors and Microsystems*, vol. 47, pp. 252–261, 2016.

[59] M. Soni, "End to end automation on cloud with build pipeline: the case for devops in insurance industry, continuous integration, continuous testing, and continuous delivery," in *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. IEEE, 2015, pp. 85–89.

[60] M. Cusumano, "Cloud computing and SaaS as new computing platforms," *Communications of the ACM*, vol. 53, no. 4, pp. 27–29, Apr. 2010.

[61] A. Benlian and T. Hess, "Opportunities and risks of software-as-a-service: Findings from a survey of IT executives," *Decision Support Systems*, vol. 52, no. 1, pp. 232–246, Dec. 2011.

[62] S. M. Greenstein, "Lock-in and the Costs of Switching Mainframe Computer Vendors: What Do Buyers See?" *Industrial and Corporate Change*, vol. 6, no. 2, pp. 247–273, 03 1997.

[63] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba, "Adoption, support, and challenges of infrastructure-as-code: Insights from industry," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 580–589.

[64] J. Opara-Martins, R. Sahandi, and F. Tian, "Critical review of vendor lock-in and its impact on adoption of cloud computing," in *International Conference on Information Society (i-Society 2014)*, Nov. 2014, pp. 92–97.

[65] ——, "Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective," *Journal of Cloud Computing*, vol. 5, no. 1, p. 4, Apr. 2016. [Online]. Available: https://doi.org/10.1186/s13677-016-0054-z

[66] E. A. C. Bittner and J. M. Leimeister, "Why Shared Understanding Matters – Engineering a Collaboration Process for Shared Understanding to Improve Collaboration Effectiveness in Heterogeneous Teams," in *2013 46th Hawaii International Conference on System Sciences*, Jan. 2013, pp. 106–114, iSSN: 1530-1605.

[67] M. Glinz and S. A. Fricker, "On Shared Understanding in Software Engineering: An Essay," *Comput. Sci.*, vol. 30, no. 3-4, pp. 363–376, Aug. 2015.

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

[69] A. Sutcliffe and P. Sawyer, "Requirements elicitation: Towards the unknown unknowns," in *2013 21st IEEE International Requirements Engineering Conference (RE)*, July 2013, pp. 92–104.

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

[71] C. Gralha, D. Damian, A. I. T. Wasserman, M. Goulão, and J. Araújo, "The Evolution of Requirements Practices in Software Startups," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: ACM, 2018, pp. 823–833, event-place: Gothenburg, Sweden.

[72] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe, "Toward data-driven requirements engineering," *IEEE Software*, vol. 33, no. 1, pp. 48–54, 2015.

[73] E. C. Groen, N. Seyff, R. Ali, F. Dalpiaz, J. Doerr, E. Guzman, M. Hosseini, J. Marco, M. Oriol, A. Perini *et al.*, "The crowd in requirements engineering: The landscape and challenges," *IEEE software*, vol. 34, no. 2, pp. 44–52, 2017.

[74] W. Behutiye, P. Karhapää, D. Costal, M. Oivo, and X. Franch, "Non-functional requirements documentation in agile software development: challenges and solution proposal," in *International conference on product-focused software process improvement*. Springer, 2017, pp. 515–522.

[75] N. Misaghian and H. Motameni, "An approach for requirements prioritization based on tensor decomposition," *Requirements Engineering*, vol. 23, no. 2, pp. 169–188, 2018.

[76] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education, 2010.

[77] J. Humble and G. Kim, *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution, 2018.

[78] J. Schwarz, A. Steffens, and H. Lichter, "Code smells in infrastructure as code," in *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*. IEEE, 2018, pp. 220–228.

[79] A. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE '19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 164–175.

[80] K. Saatkamp, U. Breitenbücher, O. Kopp, and F. Leymann, "Topology splitting and matching for multi-cloud deployments." in *CLOSER*, 2017, pp. 247–258.

[81] R. Shu, X. Gu, and W. Enck, "A study of security vulnerabilities on docker hub," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 269–280.

[82] Z. Á. Mann and A. Metzger, "Optimized cloud deployment of multi-tenant software considering data protection concerns," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2017, pp. 609–618.

[83] M. R. Roller and P. J. Lavrakas, *Applied Qualitative Research Design: A Total Quality Framework Approach*. Guilford Press, 2015.