How a Remote Software Organization Builds a Shared Understanding of NFRs

by

Laura Onyinyechi Okpara
B.Sc., Covenant University, 2017

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

How a Remote Software Organization Builds a Shared Understanding of NFRs

by

Laura Onyinyechi Okpara
B.Sc., Covenant University, 2017

Supervisory Committee

---

Dr. Daniela Damian, Supervisor
(Department of Computer Science, Uvic)

---

Dr. Adam Murray, Co-Supervisor
(Department of Computer Science, Uvic)

## ABSTRACT

Building a shared understanding of non-functional requirements (NFRs) is a known but understudied challenge in requirements engineering, primarily in organizations that adopt continuous software engineering (CSE) practices. During the peak of the COVID-19 pandemic, many CSE organizations complied with working remotely due to the imposed health restrictions; some continued with remote work while implementing business processes to facilitate team communication and productivity. In remote CSE organizations, managing NFRs becomes more challenging due to the limitations to team communication coupled with the incentive to deliver products quickly. While previous research has identified the factors that lead to a lack of shared understanding of NFRs in CSE, we still have a significant gap in understanding how CSE organizations, particularly in remote work, build a shared understanding of NFRs in their software development.

This thesis presents a study that explores how a remote CSE organization builds a shared understanding of NFRs. We conducted a six-month case study of a remote CSE organization using ethnography-informed methods and methods from grounded theory. Through thematic analysis of our qualitative data from interviews and observations, we identify some practices in building a shared understanding of NFRs, such as validating NFRs through feedback. In addition, we identified some of the impediments to building a shared understanding of NFRs in the organization, such as gaps in communication and the limited understanding of customer context.

Furthermore, we conducted member-checking interviews to validate our findings for relevance and to gain additional insights on the shared understanding of NFRs within the organization. The collaborative workspace the organization uses for remote interaction is Gather, which simulates physical workspaces, and which our findings suggest allows for informal communications instrumental for building shared understanding.

As actionable insights, we discuss our findings in light of proactive practices that represent opportunities for software organizations to invest in building a shared understanding of NFRs in their development.

# Contents

# List of Tables

# List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

**Daniela Damian and Adam Murray** for their exceptional support and guidance throughout my journey as a graduate research student. The lessons and guidance from them has been key to my growth as a researcher.

**Our partner organization** for their time and collaboration.

**The SEGAL Group** for their support, friendship and assistance throughout my journey as a graduate research student.

**My family** for their encouragement, love and patience.

**Natural Sciences and Engineering Research Council of Canada (NSERC)** for funding this research study.

*Have I not commanded you? Be strong and courageous. Do not be afraid; do not be discouraged, for the Lord your God will be with you wherever you go.*
Joshua 1:9 (NIV)

*The Lord bless you and keep you; the Lord make his face shine on you and be gracious to you; the Lord turn his face toward you and give you peace.*
Numbers 6:24-26 (NIV)

*I have seen something else under the sun: The race is not to the swift or the battle to the strong, nor does food come to the wise or wealth to the brilliant or favor to the learned; but time and chance happen to them all.*
Ecclesiastes 9:11 (NIV)

# DEDICATION

I dedicate this thesis to my Late father, Simon Ndu Okechukwu Okpara and my mother, Patricia Nwakaego Okpara. Thank you for your love, kindness and consistent support. I love you!

# Chapter 1

# Introduction

Many software organizations have adopted continuous software engineering (CSE) practices, such as continuous integration and delivery [51], to support the release of working software through automation and shorter cycle times between releases [33]. CSE is the application of tools to increase the frequency and quality of new releases of commercial-grade software [14]. The ultimate goal of continuous software engineering is to take a holistic view of a software production entity, whether this is a single software organization or an ecosystem where different organizations together deliver a final product [78].

CSE has implications for requirements engineering practices as it combines activities from agile methodologies that emphasize individuals and interactions, working software, customer collaboration, and response to rapid changes [34]. However, in CSE organizations, non-functional requirements (NFRs) are sometimes neglected and there may be a lack of formal acceptance testing for NFRs [11]. Acceptance testing is one of the main practices for requirements validation to ensure consistency, completeness and realism of requirements [24]. In addition, previous research has identified the lack of shared understanding as a significant challenge in managing NFRs in CSE [88].

Therefore, the shared understanding of NFRs is essential due to the complex, cross-cutting nature of NFRs [4] and the importance of NFRs to software projects. For example, consider an NFR such as *security* [85] [75]; e.g. if data encryption is poorly understood by developers and implemented inaccurately, a data breach could occur, exposing user information. Werner et al. describe the factors that contribute to the lack of shared understanding of NFRs, such as the fast pace of change of software requirements and lack of domain knowledge [87]. However, there is still a

lack of empirical evidence regarding how a CSE organization builds and manages a shared understanding of NFRs.

## 1.1 Motivation

During the COVID-19 pandemic, there was a global shift from in-person activities to working remotely [12]. Many organizations had to adapt processes and systems to support team productivity, through virtual communication tools, social engagements, and peer support [66]. Working remotely creates further barriers to building a shared understanding of NFRs because conveying information through non-verbal communication is difficult in remote environments [56]. Some of the barriers to remote collaboration are geographical distance, differences in context and heavy reliance on communication technologies [36].

With the significant shift of software organizations working remotely, the motivation for this study stems from the need to explore how remote CSE software organizations build a shared understanding of NFRs to potentially minimize the costs that may result from a lack of shared understanding of NFRs; such as rework [87].

The primary goal of this study is to investigate how remote CSE software organizations build and maintain a shared understanding of NFRs. The secondary goal of this study is to provide additional insight into the impediments or challenges to a shared understanding of NFRs in CSE. Two main research questions guided our study:

**RQ1:** How does a *remote* software organization that adopts CSE practices reach a shared understanding of NFRs?

**RQ2:** What are the impediments to the shared understanding of NFRs in a *remote* software organization that adopts CSE practices?

## 1.2 Methodology

Our case study uses ethnography-informed methods [6] and methods from grounded theory [47] to study how a remote software organization, Alpha (fictitious name to protect confidentiality), attempts to build shared understanding in their product development. The ethnography-informed methods we use include observing participants at Alpha during their everyday work life and collecting different types of data related

to how they build shared understanding while keeping an open mind for new discoveries and possibilities [76]. In our study, we refer to the methods and components of grounded theory as "grounded analysis".

In addition, we conducted interviews after months of observations, following grounded analysis, by revising the interview questions twice during the weeks of the scheduled interviews. In an ethnographic study, the researcher usually gains more understanding of the subject area over time to identify and focus on more interesting or relevant aspects of the study [76]. Our study was conducted over a period of six months.

For our data analysis, we use grounded analysis, specifically open and selective coding approaches incorporated with memoing and constant comparison, to derive the themes that form the findings of our case study. In Chapter 3, we describe our methodology in detail.

## 1.3 Research Contributions

This study provides four meaningful contributions for both researchers and practitioners. This study

1. presents a detailed explanation of the practices for building a shared understanding of NFRs observed at Alpha,

2. presents a detailed explanation of the impediments to building a shared understanding of NFRs observed at Alpha,

3. describes the effects of remote interaction on shared understanding in CSE and how Alpha mitigates some of these effects, and

4. presents an exploration of shared understanding of NFRs in CSE at Alpha and how similar organizations can be more proactive in building a shared understanding of NFRs.

## 1.4 Research Publications

Our research method and contributions culminated in the following publication(s):

1. Laura Okpara, Colin Werner, Adam Murray and Daniela Damian. "A Case Study of Building Shared Understanding of NFRs in a Remote Software Organization," 2022 IEEE 30th International Requirements Engineering Conference (RE' 2022), Melbourne, Victoria Australia, 2022.

## 1.5 Thesis Outline

This thesis is organized as follows:

**Chapter 1: Introduction** provides the motivation for the research, as well as the research questions, methodology and contributions.

**Chapter 2: Background and Related Work** provides the context of the research and existing work in related areas of this work.

**Chapter 3: Methodology** describes the research process. Specifically, steps taken to collect and analyze relevant data and validate the results.

**Chapter 4: Research Findings** provides the findings from the research process. We describe the practices for building a shared understanding of NFRs observed at Alpha, as well as, the impediments to building a shared understanding of NFRs. In addition, we describe the practices for improving remote collaboration when building a shared understanding of NFRs.

**Chapter 5: Discussion and Implications** describes the effects of remote interaction and CSE on shared understanding. We also describe the significance of the practices for building a shared understanding of NFRs and what it means for practitioners and researchers in requirements engineering.

**Chapter 6: Threats to Validity** describes the limitations as well as the threats to validity of this study.

**Chapter 7: Future Work** provides suggestions for future work and summarizes this study.

**Appendix A** provides the recruitment documents used during our research study. This includes the email to our partner organization, invitation email to participants and participant consent form.

**Appendix B** provides the interview questions and codebook derived from the first phase of interviews conducted with Alpha.

**Appendix C** provides the interview questions and codebook derived from the member checking interviews conducted with Alpha.

**Appendix D** provides our publication to the 30th International Conference on Requirements Engineering, 2022.

# Chapter 2

# Background and Related Work

In this chapter, we first provide an overview and definitions of the topics that form the context of our research, namely, shared understanding (SU), non-functional requirements (NFRs), and continuous software engineering (CSE). Thereafter, we summarize relevant research by exploring the shared understanding of NFRs in practice and how remote collaboration affects the shared understanding of NFRs in CSE. Finally, we discuss the current state of research on the shared understanding of NFRs in CSE.

## 2.1 Background

### 2.1.1 Shared Understanding (SU)

Shared understanding cuts across several fields, disciplines, and contexts, such as education [72], medical science [42], design science [15], or engineering [40]. Several definitions of shared understanding exist in literature; one study describes shared understanding as, "the ability of multiple agents to coordinate their behaviours with respect to each other to support the realization of common goals or objectives" [13]. Hinds conceptualizes shared understanding as a collective way of organizing relevant knowledge and a means for team members to anticipate and predict the behaviors of their peers or group [44]. These definitions focus on people and how they communicate to reach a common goal or objective.

Previous research suggests that shared understanding among people exists in two forms: implicit and explicit [40]. There is implicit shared understanding when people have a mutual understanding of unspecified facts or assumptions; whereas there is explicit shared understanding when people have a mutual understanding of specifica-

tions such as documents. In addition, software teams rely heavily on implicit shared understanding because stories and system metaphors [40] provide a general direction on the product to be built.

Furthermore, research extends the concept of shared understanding to include shared mental models. The theory of shared mental models is explained as *the knowledge structures held by members of a team that enable them to form accurate explanations and expectations for tasks and in turn, coordinate their actions and adapt their behaviour to demands of the task and other team members* [20]. The knowledge described does not only refer to the knowledge of the organization or team, but also the ability of team members to develop an understanding of how they can make contributions and act based on that knowledge [48].

### 2.1.2 Non-functional Requirements (NFRs)

Non-functional requirements (NFRs) have been described and classified in several ways. Prior research defines an NFR as a property, or quality, that the product must have, such as an appearance, or accuracy [16]. For example, in one study, researchers report that software architects describe the most important types of NFRs such as usability [4]. This is because these NFRs represented the explicit needs of their customers, e.g, clients may need to be satisfied with the user interface of their software products. In addition, NFRs and product quality attributes are closely related as quality attributes describe the degree of how stakeholders' needs are satisfied [69], which is critical to the success of a project [39].

Other NFRs such as security and privacy are vital for software organizations due to global and local privacy regulations that protect users [60]. For instance, in Canada, the Personal Information Protection and Electronic Documents Act (PIPEDA) [83], offers protection for personal information about an identifiable individual. This means that in order for software organizations to comply with this act, they must ensure that user data is protected sufficiently.

While previous research has described a framework for integrating NFRs into the software development process [67], there is some empirical evidence that organizations may label functional requirements as NFRs [30], and so NFRs are still under-specified and described at different levels of abstraction [30].

### 2.1.3 Continuous Software Engineering (CSE)

CSE increases the need for an effective flow between customer needs and the rapid delivery of a product or service [33] through the automated release of working software in short release cycles. While this concept is already established through the agile manifesto [34], CSE is a more holistic approach as it includes *continuous activities* that extend these practices such as continuous verification, improvement, and innovation [33].

Continuous verification refers to adopting software verification activities such as formal methods and inspections throughout the software development process rather than waiting for a testing phase after the full implementation of the software product [33]. Continuous integration, as CSE emphasizes, provides quick feedback that may support the frequent release of working software, ensuring that potentially failure-inducing problems are identified and resolved as quickly as possible [33].

In a recent study, Johanssen et al. found that practitioners have five different perspectives on CSE namely, as a tool, methodology, developer-driven, software life-cycle and product management perspective [51]. For example, from the methodology perspective, the researchers observed that *practitioners define CSE from a methodological perspective that aims for short iterations during software evolution.* Their study introduces the *Eye of CSE* to serve as a tool that helps practitioners recognize CSE elements and categories based on their proximity within the eye and the position of the CSE element [51]. The model Eye of CSE consists of nine CSE categories and 33 CSE elements, as described further in the study [50].

## 2.2 Related Work

### 2.2.1 Shared Understanding in Remote Collaboration

In remote collaboration, developers and designers face additional challenges with building a shared understanding such as the limitations to the way knowledge is shared among the developers [55]. In many remote software organizations, software developers are geographically distributed across several countries or regions which creates a barrier to communication limited by differences in culture or time-zones. In a study, Yvonne Hsieh describes the impact of culture on geographically distributed remote teams as the differences in the way developers approach communication and problem-solving and the absence of informal communication i.e “water fountain talk”[48].

Research evidence in requirements engineering suggests that inadequate communication contributes to a lack of shared understanding [87]. Prior research suggests that in software organizations, informal communication is essential for understanding and communicating about stakeholder values and needs [10]. Additionally, daily informal communication allows team members to develop working relationships and encourages a better flow of information [2].

Informal communication is interactive and includes unplanned interactions that occur in the midst of daily activities [84] [27]. Kraut et al. discuss informal communication as based on the degree of pre-planning, therefore, informal communication can be scheduled, intended, opportunistic, or spontaneous [57]. Previous studies suggest that informal communication is one of the best ways to build trust among people in groups to maintain productivity, especially in software-intensive organizations [43].

Software developers spend more than half of their work day on activities that include collaboration or interactions with other developers [29]. When the software team does not share a physical work environment, information exchange between them becomes challenging without an effective technological support [28]. Research evidence suggests that having a virtual shared space in remote software organizations is important for collaboration on software projects [8]. For example, Dullemond et al. introduced a tool for virtual open conversation spaces, Communico, with the main requirement to support open conversations where people can see other ongoing conversations that do not include them [28]. The researchers conducted interviews with users to evaluate Communico. They found that users felt more connected when joining conversations using this tool. The users also liked how easily they could actively join these conversations without waiting to be invited in. However, the researchers found that Communico has some limitations. First, users are not notified when new conversations start. Second, the users cannot see other users watching the conversation. In addition, in Communico, all conversations among users are public [28]; however, in a traditional work environment, software developers often have private conversations behind closed doors or in close channels.

To improve remote collaboration in software teams, there is a need for research and tools on virtual shared spaces that closely simulates the in-person work environment. This is a gap that we intend to fill by offering empirical evidence on *how* a remote software organization manages communication of NFRs in the absence of traditional in-person informal communication.

### 2.2.2 Managing NFRs in Continuous Software Engineering

Managing NFRs in CSE remains an understudied area in requirements engineering. Several studies focus on testing or verifying NFRs or continuous activities, but not both. However, a recent study by Werner et al. describes how three software organizations manage NFRs in their continuous environment [88]. Their research suggests that one of the practices for managing NFRs is through defining metrics to help validate the NFR. For example, one of the organizations in their study define quantitative metrics for PERFORMANCE, such as response time and caching ratio and monitors these metrics using a third party tool that sends notifications when issues occur.

Nevertheless, a previous empirical study suggests that managing NFRs can be challenging to software organizations for varying reasons, such as sparse documentation of NFRs or when documentation may be imprecise [4], along with the lack of consensus on how to sufficiently document and communicate NFRs [39]. Research suggests that NFRs have a representation problem and when NFRs are not described or represented properly, they can be misinterpreted as FRs [39]. NFRs such as *security* and *privacy* are important to software projects, but often neglected [85] and may be specified too late resulting in system vulnerabilities [75].

NFRs are important to the success of software projects; however, research suggests that there is still some conflict in how software organizations manage NFRs [64]. NFRs may also tend to contradict with one another due to the cross-cutting nature of many NFRs or the complex relationships among NFRs. In a research study by Mairiza et al., they found that NFR conflicts may present as absolute conflicts or relative conflicts [62]. Relative conflicts suggest that the NFRs may conflict in certain scenarios but not always. For instance, the accuracy or robustness of a system may be described as performance requirements by some developers.

NFRs are often neglected in requirements engineering due to the difficulties in eliciting NFRs, as NFRs have cross-cutting concerns with FRs [85]. At Alpha, the software development team elicits NFRs mainly through product demos or sharing mockups with customers then documenting the feedback received. Additionally, the software development team at Alpha do not separate FRs from NFRs in requirements specification documents.

In CSE, continuous activities such as continuous integration (CI), continuous testing and continuous verification enable software developers to continuously test and verify functional and non-functional requirements [14]. However, in CSE, NFR test-

ing may require many software tools to be integrated into the CI environment leading to an unstable CI environment [89]. NFR testing tools are hard to integrate in a CI environment because there may not be an appropriate visualised UI interface, and this contributes to creating an unstable CI environment. In addition, some NFRs are hard to test and automate in a continuous environment because many NFRs cannot be broken down into basic functions that can be fulfilled by a component in a product [89] e.g. USABILITY.

The main challenge is to understand the complex nature of NFRs and to create practices to manage the shared understanding of NFRs [63]. This is the gap that we intend to fill in this study by exploring practices to build a shared understanding of NFRs.

### 2.2.3 Current Challenges with the SU of NFRs in remote CSE

There is still little research on the shared understanding of NFRs in CSE. Werner et al. studied shared understanding of NFRs in CSE within three organizations [87]. They traced the effects of the lack of shared understanding in the form of rework in software projects, and they found that managing NFRs in CSE comes at a cost. They found that the lack of shared understanding of NFRs was due to several factors, such as the fast pace of change of software requirements that may lead to a little emphasis on NFRs due to pressure to release a product and sometimes a lack of testing of the NFRs. Other factors were inadequate communication, and a lack of domain knowledge. They recommended practices to avoid a lack of a shared understanding of NFRs: improved communication and documentation, and shared development standards.

Similarly, previous research reports the difficulties with automating NFRs and the problem of prioritizing functional requirements (FR) over NFRs [88]. They found that NFRs such as usability are difficult to verify through automated tests, so a CSE organization may rely on manual verification; this can eventually lead to a resource bottleneck. They found that in resource-constrained environments, a CSE organization may make NFR trade-offs with hopes of a remedy through the product's success later in the future. However, these NFR trade-offs often led to technical debts, due to the fast pace of change of software requirements in CSE organizations.

A study by Kniel et al, explores how software designers perceive shared understanding when collaborating remotely [56]. The researchers identified the perceived enablers of shared understanding in remote software teams, such as team spirit, shared experience, transparency and project management. The research evidence also suggests that there are barriers to shared understanding in remote software teams, like challenges in information sharing, cultural differences, and overruling decisions. In our study, we discuss some of the enablers of shared understanding from Kniel et al [56] and evaluate the extent to which our research findings support or contradict.

Furthermore, shared understanding is beneficial to CSE projects as it can promote successful collaboration [22], especially in remote software organizations with greater collaboration challenges. When shared understanding exists, a team member can predict others' behaviours, and there is an increase in team motivation and less conflict or mistrust among the team [22].

In our study, we bring further evidence towards how shared understanding of NFRs is built in remote software organizations that adopt CSE practices.

# Chapter 3

# Methodology

In the following sections, we describe our research approach and the methods we used.

## 3.1 Research Methodology

For this research, we used qualitative methods, relying on ethnographic-informed methods and grounded theory in a single case study. For this thesis, we conducted participant observations and interviews with the participants. We analyzed the data from the observations and interviews using grounded analysis. Our interpretation of the results form the findings of this study. Figure 3.1 represents the elements of our research methodology.

### 3.1.1 Ethnography

Ethnography is a qualitative research methodology commonly used in studies about people or cultures. However, it is now largely adopted across several disciplines, including software engineering. For instance, Sharp et al. uses ethnography to create a framework to study how software orgrounded analysisnization practitioners apply software quality procedures [77]. Their main data collection methods were through participant observations and informal interviews, which are largely based on ethnography.

Sharp et al. describe four major features of ethnographic research [76]. In their study, they refer to research participants in an ethnographic study as "informants". First, ethnography focuses on the informant's point of view and this usually leads to a longer research duration. Second, in ethnographic studies, the researchers are

interested in the daily ordinary activities of the informants and so researchers often collect data about different aspects of the informant's work, mainly though observations. Third, in ethnographic studies, the researcher often interprets and analyses the data collected. Lastly, the results from an ethnographic study is often detailed and comprehensive as it aims to communicate the full picture.

Another example of an ethnographic study in computer science is the work by Nahoko Kameo., who explores the culture of uncertainty in software engineering teams through a year of field observations [53]. The researcher studied a software orgrounded analysisnization that recently introduced an agile methodology called "Scrum". The researcher observed the software engineering work and collaboration at the orgrounded analysisnization and conducted some interviews with managers to grounded analysisin additional insights. Through the field observations and interviews, the researcher found evidence on the process through which orgrounded analysisnizational memory influences the ongoing orgrounded analysisnizational life. The ethnographic approach enabled the researcher to conduct continuous observations to enrich the results from their study.

In ethnographic studies, observation is the key form of data collection where the researcher will either participate in the informants' activities, or observe without participation [76]. We chose methods from ethnography because it informed our participant observation, interviews for data collection and the comprehensive documentation of our data and findings.

### 3.1.2 Grounded Theory

Grounded analysis is a research method that allows researchers to systematically develop a grounded theory (GT) that is based on research evidence [45]. Glaser and Strauss introduced grounded analysis techniques through a book called *The Discovery of Grounded Theory* [38]. Grounded analysis has been used to explore the human and social aspects of software engineering because it is useful when exploring understudied research areas and it allows the generation of concepts and categories that lead to the main themes in the research findings [47]. For example, Adolph et al. used grounded analysis to study how people manage the software development process [1]. They explored how people interacted with each other to solve problems in software development. Another example of grounded analysis is the work of Terpstra et al., who sought to discover how agile software developers understand security requirements in

agile projects [82]. The iterative process of grounded analysis enabled the researchers to develop a model that describes the questions practitioners ask themselves when trying to understand the security requirements challenges in their agile projects.

One of the key components of grounded analysis is that data analysis occurs in near constant comparison. Throughout the study, the researcher consistently compares data, memos, codes and categories [81]. The researcher evolves the interpretation of the data and categories until theoretical saturation. Theoretical saturation refers to when the researcher's interpretation is supported by the data, and new data does not prompt additional revisions or interpretations of the results [81].

In addition, in the grounded analysis process, the researcher writes theoretical notes as memos to elaborate categories as they emerge [81] and describe the relationship between categories [47]. Memoing allows the researcher to make assumptions and write down their interpretation of the data. Another key component of grounded analysis is theoretical sorting, a process that involves examining, arranging and re-arranging memos to enable theoretical structures or ideas to emerge [45]. Glaser enforces the idea that grounded analysis has a focus on the emergence of research questions, codes or categories [81].

We chose grounded analysis for our research method because grounded analysis allows researchers to explore study areas with limited research such as the shared understanding of NFRs in CSE. Additionally, memoing allowed us to explore our interpretation of the data to find patterns and relationships that emerge from the codes and categories.

This thesis explores "how" software developers build a shared understanding of NFRs by describing the experiences of the software developers with NFRs in a CSE environment.

Grounded analysis and ethnography have been compared and combined in prior research studies in different ways. One research study uses the combined methods to study how children interact with children mobile libraries [9]. The researchers chose ethnography to study the children without intruding in their lives, while hearing their perceptions and interpreting those views to adults. The researchers chose grounded analysis to compare children mobile libraries across the region in an attempt to build a theory that emerges from the data. Another study combines the two methods to study computational communication as an interdisciplinary research study [70].

Concerning the differences between the methods, in grounded analysis, Glaser

recommends that researchers should not conduct the literature review before conducting the fieldwork [81], to avoid constrained coding and memoing. Whereas, in ethnography, researchers can perform literature review before conducting the study [76]. Concerning similarities, observations are common to both grounded analysis and ethnography because researchers perform their studies in the natural everyday context of the topic or concept [47][76]. Other studies compare ethnography and grounded analysis to establish probable use for the methodologies, the main differences and similarities between them [68] [3].

We chose methods from both ethnography and grounded analysis because they enabled us to conduct participant observations and interviews, and ground our research results in the theoretical structures and ideas emerging from our data.

## 3.2 Research Setting

We elaborate a remote organization, Alpha, that practices CSE. Alpha is a Canada-based software organization that develops loyalty, rewards, and referral programs for businesses, as their primary revenue source. Alpha has operated for 10 years with 29 global employees across at-least four timezones. Alpha is primarily client-focused, with most of its products having heavy visual components, including third-party platforms and storing client information.

At Alpha, NFRs such as performance, usability, security, and maintainability are essential for their software products to meet their stakeholders' needs. The software engineering team at Alpha includes project leads (manager or developer), solutions architects, front-end and back-end software developers, user experience (UI/UX) designers/developers, and software integration engineers.

During the COVID-19 pandemic, Alpha shifted to remote work to comply with the health restrictions imposed on businesses within the region and country. Having adapted its business operations and approach, Alpha stayed fully remote for over two years. Alpha used Gather, a video chat platform designed to make virtual interactions more human.

Subsection 3.2.1 describes observations from our study on how the software development team at Alpha communicates and shares knowledge while working remotely. Similarly, Subsection 3.2.2 describes our observations of Alpha's software and requirements engineering process, including how Alpha validates NFRs in their CSE environment.

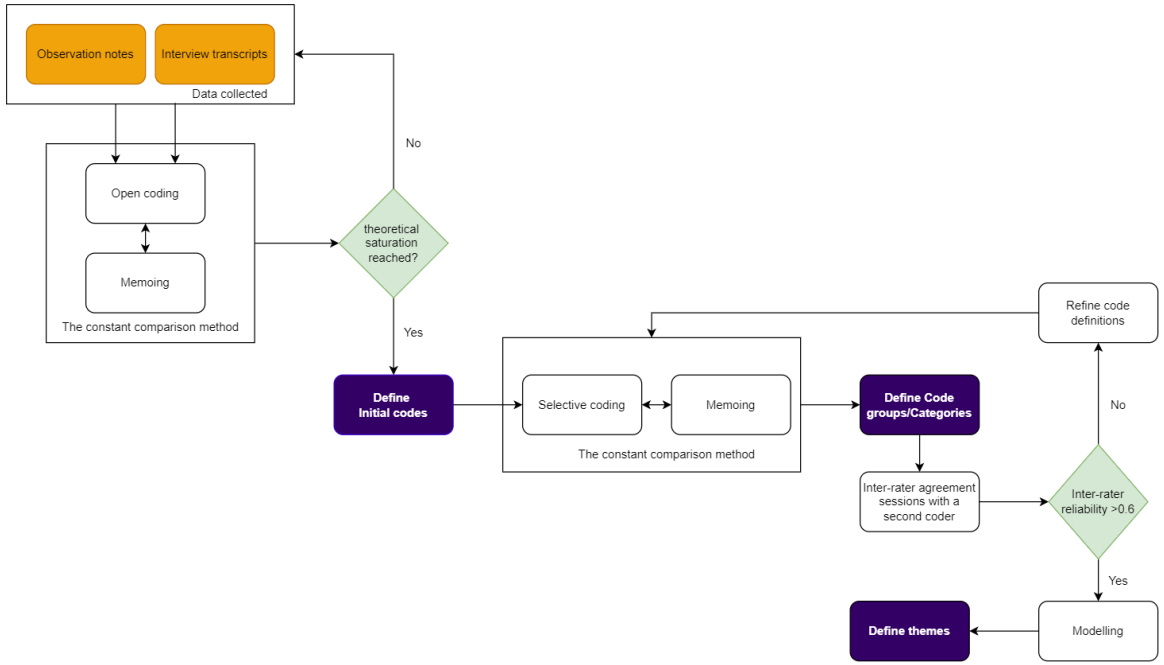



Figure 3.1: This chart shows we used various data collection and analysis techniques to develop the findings of this work.

### 3.2.1 Alpha's communication methods and tools

Similar to many other organizations during the COVID-19 pandemic, Alpha had to pivot to work remotely. Alpha initially used video conferencing tools such as Zoom; however, it soon discovered that Zoom did not create connectedness within the software team, e.g. A manager explained

> "we started out like that when the pandemic first hit, we all went remote. Initially, it was so disconnected. No one felt connected to anyone else. It was so difficult to engage people in random conversation".

Alpha then switched to using Gather [35], a virtual interaction tool that creates a work environment similar to a physical workspace, including desk spaces, meeting rooms, and avatars that represent the team members. Gather is fully customizable and allows Alpha to interact virtually as much as they would do in a physical space: whenever an avatar (team member) "walk" near another team member's avatar, a video call would be initiated in an easy way, to simulate the informal communication available in physical co-location, e.g A manager states

> "Gather has made it possible for us as a remote team to have informal conversations. You literally just walk your little character up to someone else's little character, and you're talking to them".

During meetings, Alpha employees "walk" their avatar to the virtual meeting room in Gather and can choose to sit at the table. A group video call on Gather would begin as soon as they are near each other. Alpha's employees can also create private spaces "bubbles" with each other while in group meetings for one-on-one conversations when necessary, creating the feeling of being in the same space with each other. Alpha uses Gather to host social events, such as Christmas parties, encouraging team members to interact with each other beyond their daily work.

Alpha also uses Slack [79] for asynchronous messaging and to provide and receive frequent updates relevant to daily activities. Alpha uses web documentation tools to communicate and document requirements such as Coda [19] to review backlogs, view the status of previous or ongoing tasks, and update tasks seamlessly throughout the life cycle of projects.

### 3.2.2 Alpha software and requirements engineering processes

Alpha adopts CSE practices such as continuous verification and continuous testing based on stakeholder needs. Empirical evidence suggests an organization that adopts CSE may focus on certain CSE practices, such as continuous integration for several reasons to suit their business and stakeholder needs [31]. The software engineering team at Alpha uses continuous delivery to frequently deliver working software in short cycles, welcoming changing requirements to meet the stakeholders' needs.

In addition, Alpha implements a people-driven, autonomous approach incorporating tribes and guilds [80]. A tribe describes a group of people who work on similar feature areas of the software products; they coordinate and align themselves through the tribe leader, usually a senior member of the tribe. Tribe members have the software engineering skills to collaborate and engineer working software features from "end-to-end". A guild describes a group of people with similar interests who have similar roles or perform similar tasks. Guild members convene to share knowledge, support each other and share processes to create a community that fosters learning and growth in specific areas. Belonging to a guild is usually optional, and a guild may have a guild leader who is a more experienced senior guild member.

Before starting a new project, a project kick-off meeting first occurs that includes stakeholders, such as project engineers, success representatives, customers, or upper-level management. The meeting defines some of the high-level software requirements, including NFRs, and the project scope through user/data flows, product designs to understand the software product to be built. The project evolves with the delivery of working usable software in short cycles by continuously verifying project requirements with stakeholders throughout the software development process, e.g. A developer describes this as

> "to ensure that the right thing has been built and it has been built right".

The project lead initiates frequent feedback cycles such as QA, weekly or bi-weekly meetings with the stakeholders involved in the project; the feedback from these meetings can be change requests, new requirements or software bugs discovered. For instance, At Alpha, when applications errors are not handled, it affects some important NFRs, such as the user experience, application reliability, and developers' ability to maintain the codebase easily (maintainability). The error handling issue is usually flagged as a bug and assigned to a developer working on the project. After

the issue is resolved, it is reviewed and verified at a later time. The rapid feedback cycles continue and the project continues to evolve until all requirement requests have been implemented, all of the stakeholders are satisfied with the product built or the project is discontinued.

At Alpha, developers document software requirements as executable specifications using Cucumber [21], a test tool and framework that follows the behavior-driven design methodology (BDD) [59] to support the user and behavioural testing. At Alpha, developers have some freedom and flexibility to implement the defined and documented software requirements, especially the NFRs. However, the software developers must collaborate with the project lead to remove ambiguity from the software requirements and create a software architecture that allows all components to work together.

The software engineering team at Alpha validates important NFRs such as performance, usability and maintainability. For performance requirements, Alpha monitors the performance of an API or application through an application performance monitoring tool, DataDog [23], that enables them to spool and visualize metrics such as error rates, count of application instances and average response times for API calls. Alpha validates usability mainly through requesting feedback during demos with internal and external stakeholders. Alpha also uses behavioural testing via storybooks or stencil books to manually verify the ability to reach specific pre-defined goals through the user interface. Alpha also measures an EMOTIONAL RESPONSE SCORE during user acceptance testing with stakeholders by gauging their reactions during demos or presentations. Alpha validates other NFRs such as maintainability and testability during code reviews, deep-dive meetings or daily download meetings where a senior developer can review the written code to identify issues. In addition, Alpha uses automated test coverage tools to validate that the unit tests written for an NFR such as security covers all possible functions and test cases, including authentication or authorization.

## 3.3 Phase 1: Data Collection

In this section, we describe the data collection methods used in our study. Section 3.4 describes how we analyzed the collected data. Our data collection techniques were through observations and interviews and are described in details below.

### 3.3.1 Ethical Considerations

As part of the design process for the data collection, we submitted a human research ethics board (HREB) application, seeking approval for our research protocol. Our ethics application influenced our data collection methods and process, data storage, data disposal and results publication.

Our study was approved by the HREB, with the reference number 21-0656. As part of our ethics application, we submitted recruitment documents and details of our study protocol. After we received approval, we shared the recruitment email with Alpha, described in Appendix A.1.

Alpha on-boarded the on-site researcher by providing a work email, providing a dedicated computer, and granting access through invitation to their collaboration tools and platforms such as Gather, Slack, Coda, Google Drive [18] and GitHub [37].

To recruit participants for our observations and interviews, we shared an invitation by email, described in Appendix A.2. Each participant was required to provide written consent to participate in our study, the consent form we used is described in Appendix A.3. We informed our participants that participation is voluntary, anonymous and open to withdrawal at any time during and after the study period. The ethics package for our study, including interview questions is attached in Appendices A, B and C.

### 3.3.2 Participant Observation

For six months, the thesis author was embedded at Alpha to learn about the products, processes, and business. Through many informal conversations with the software engineering team, we intended to understand how the team shared understanding, knowledge and how they communicate and document requirements.

The on-site researcher observed the participants and their interactions during regular work activities and meetings with the software team and stakeholders, including customers and upper-level management. Alpha granted the researcher access to project documentation and repositories, which informed some of the researcher's observation notes. The researcher stored a collection of notes made during the observations and included them as a part of the memos used in the thematic analysis process. Figure 3.2 shows an example of the researcher's observation note. The researcher dedicated an average of 30 hours each week within Alpha for research, data collection and analysis during the first three months of our study. For the last three

months, the researcher dedicated an average of 15 hours each week within Alpha to verify and make conclusions from the member checking interviews and observation notes.

By engaging in informal conversations with employees and reviewing project documentation, the researcher learned about the roles and experiences of the employees and developed a working relationship with the team to help design the interview questions, describe the study setting adequately, and inform the interpretation of interview responses.

### 3.3.3 Interviews

Eleven semi-structured interviews were conducted via Zoom during the third month of the study. Each interview time varied between 30 and 45 minutes and was fully transcribed. The interviewees had various roles with varying years of experience, from a couple of months to fifteen years of experience. To protect the anonymity of our participants, we label developer/designer roles and team leads as developers and other remaining management roles as managers, as seen in Table 3.1.

| **Role** | **Work Area** | **Exp. in Org.** | **Overall Exp.** |
|---|---|---|---|
| Developer | Visual Dev | <1 yr | <1 yr |
| Manager | Product | <5 yr | <5 yr |
| Developer | Visual Dev | <1 yr | <1 yr |
| Developer | Project Engr | <1 yr | <1 yr |
| Manager | Executive | <10 yr | <10 yr |
| Developer | Visual Dev | <1 yr | <1 yr |
| Manager | Project/Product | <5 yr | <10 yr |
| Manager | Infrastructure | <5 yr | <20 yr |
| Developer | Infrastructure | <1 yr | <3 yr |
| Developer | Infrastructure | <3 yr | <3 yr |
| Developer | Visual Dev | <3 yr | <3 yr |

Table 3.1: A description of the interview participants based on years of experience and working product area

Following grounded analysis, we iteratively developed the interview questions based on emerging patterns that helped us to establish an area of focus or interest [47]. We defined our initial set of interview questions through exploring the structure of interview questions in previous similar projects and relying on our observation notes for research context. The interviews focused on highlighting how shared understanding of NFRs is established and what challenges may limit the shared understanding

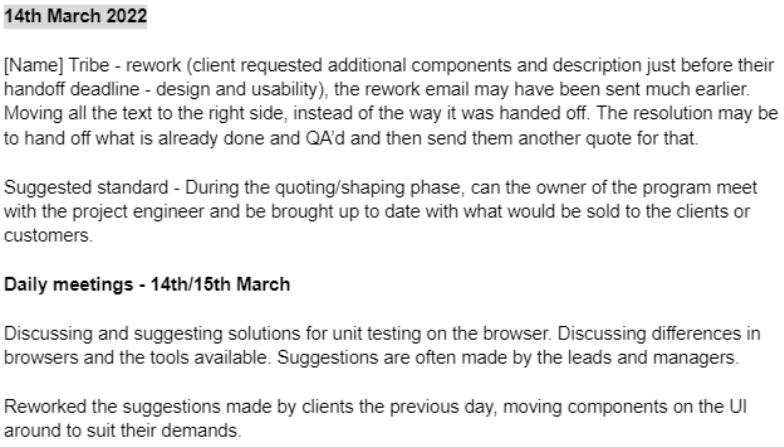
of NFRs. We used semi-structured open-ended questions to allow participants to describe their experiences. We sought each participant's view on shared understanding and how Alpha may reach a shared understanding of NFRs. The interviews were audio-recorded with consent; we informed our participants that their consent is open to withdrawal at any time. Appendix B.1 describes the full set of interview questions used in this phase of our study.

## 3.4 Phase 2: Data Analysis

This section presents the steps taken to analyze the data from the interviews and observations. The findings from this data are addressed in Chapter 4. The subsections below outline the procedures and provides literature on the specific methods we used.

### 3.4.1 Review of Observation Notes

The observation data collected was analyzed by closely examining the data to identify patterns and data relevant to our study, such as observation notes related to a response from an interview session. We used this data to describe our study setting and design the research questions for the interviews. We also used the data from our observations to interpret the interview findings by understanding and describing Alpha's context and discussing our study's implications. Figure 3.2 shows an example of our observation notes, as documented on the researcher's computer.



**14th March 2022**

[Name] Tribe - rework (client requested additional components and description just before their handoff deadline - design and usability), the rework email may have been sent much earlier. Moving all the text to the right side, instead of the way it was handed off. The resolution may be to hand off what is already done and QA'd and then send them another quote for that.

Suggested standard - During the quoting/shaping phase, can the owner of the program meet with the project engineer and be brought up to date with what would be sold to the clients or customers.

**Daily meetings - 14th/15th March**

Discussing and suggesting solutions for unit testing on the browser. Discussing differences in browsers and the tools available. Suggestions are often made by the leads and managers.

Reworked the suggestions made by clients the previous day, moving components on the UI around to suit their demands.

Figure 3.2: A screenshot of an example of our observation notes

### 3.4.2 Coding Process

Data analysis involved fully transcribing audio recordings from interview sessions and creating insights using the information obtained through observations. The interview transcripts were analyzed to identify main themes through thematic analysis [54], following the grounded analysis method [47]. We used a licensed qualitative research software called ATLAS.ti [7] for our coding process.

We used transcription software to transcribe the audio recordings from the interviews, and one of the authors manually validated that the audio recordings matched the transcripts produced. We started by familiarizing ourselves with all of the data collected by actively reading through the data set: the interview transcripts and observation notes.

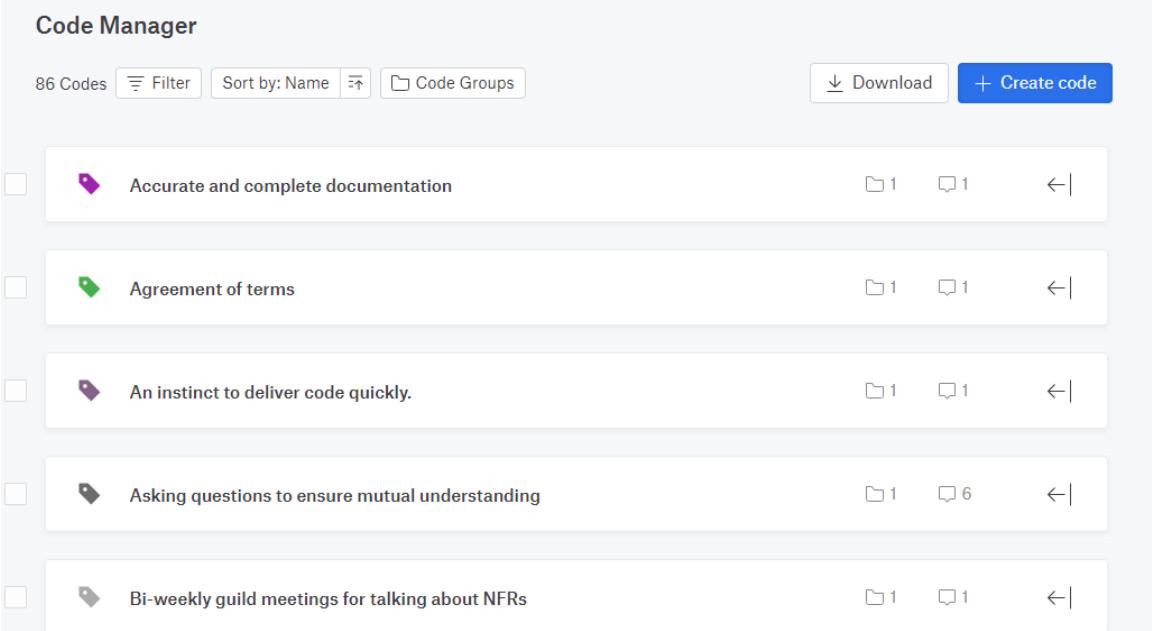



Figure 3.3: A screenshot of the open coding process - initial codes for the first few interview transcripts, using ATLAS.ti [7]

The next step in the data analysis was to generate initial codes through open coding. We did this by noting connections between data items and other relevant data for three interview transcripts [17]. Figure 3.3 shows a screenshot of the sample of initial set of codes we derived from the first three interview transcripts. Throughout the coding process, we made notes as memos using a component of grounded analysis called memoing, further described in subsection 3.4.3. In addition, we used the constant comparison component of grounded analysis to iteratively explore our findings

as they evolved.

After the first phase of coding, we used selective coding from grounded analysis to infer code groups or categories as they emerge from the first list of codes and the data. Figure 3.4 displays a sample of the resulting code groups at this stage. At this stage, two independent coders met to discuss each code and decide on whether the code applied to the categories and the examples [54]. Agreement sessions were used to unify coding strategies , and to establish the reliability of the codes. We calculated the inter-rater reliability agreement until we reached substantial inter-rater reliability >0.60, using the Cohen Kappa coefficient for measuring observer agreement for categorical data [58].


Figure 3.4: A screenshot of code categories after a few iterations of open and selective coding, using ATLAS.ti [7]

In summary, we used the "constant comparison" process with the open and selective coding methods from grounded analysis [47] to identify code categories [54] across several codes from the interviews, and by linking them to the observation notes.

Following the coding of the first three transcripts, we also use "constant comparison" from grounded analysis with open and selective coding to analyze and code the additional eight interview transcripts. Similarly, two independent researchers coded and compared notes to reach a substantial inter-rater reliability >0.60, using the Cohen Kappa coefficient for measuring observer agreement for categorical data [58].

Throughout the coding process, we inferred categories that emerge from the codes based on codes or patterns of data that reoccur. From the categories, we recognized the emergence of the core categories from which we modelled the main themes of the research. During the modelling process, we used the concept of theoretical sorting

to arrange or re-arrange the emerging core categories and memos in a way to outline themes that they represent and their relationship with other categories. Appendix B.2 describes our full codebook for the first phase of interviews.

From our codes, we also identified the triggers that led Alpha to build a shared understanding of NFRs, as suggested by Werner et al. [87]. Werner describes *triggers* as events that encourage an organization to invest in building a shared understanding [86]. We discuss the triggers in Chapter 5.

### 3.4.3 Memoing

Memoing is the technique of writing notes to elaborate code categories as they emerge, describe the relationship between codes and categories, and identify additional gaps in the research [81]. Memoing enhances the researcher's understanding of the research data as it provides an opportunity for the researcher to think retrospectively and write down their reflections about the emerging codes and categories [46].

Throughout the data analysis, we created memos to describe the relationships and patterns observed in the data and any additional questions that may be explored. For example, Figure 3.5 shows an example of our memoing process, where we describe our interpretations of an observation.

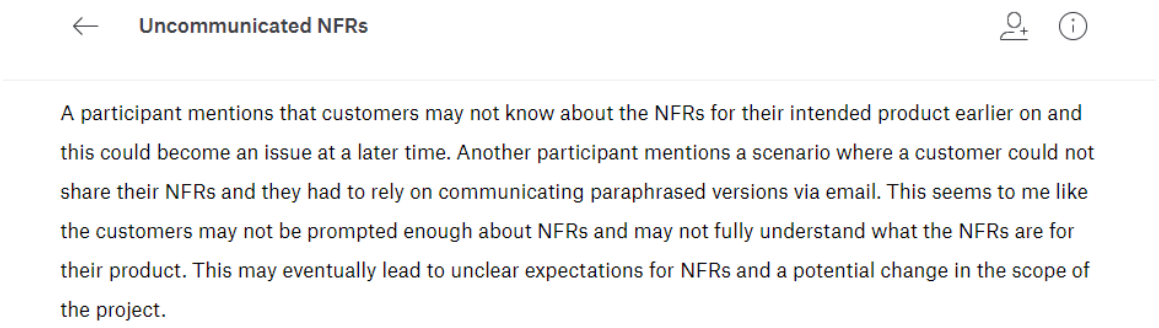



Figure 3.5: An example of the memoing for a single code using ATLAS.ti [7]. By using ATLAS.ti, the researchers can create codes, code groups and write memos to describe their interpretation of the code examples. Here, the coder explains their thoughts about the code *Uncommunicated NFRs*

.

## 3.5 Phase 3: Research Validation

To increase the credibility of our research findings, we performed member checking with the research participants through semi-structured validation interviews. We designed the interview questions based on the research findings to evaluate how much our participants agreed or disagreed with our findings. The secondary goal of the member checking process was to identify additional information that improves the richness of our research results.

### 3.5.1 Member Checking Interviews

Ten member-checking interviews were conducted via Zoom during the fifth month of the study. Each interview time varied between 30 and 45 minutes and was fully transcribed. We performed the member checking interviews with the same participants from the first interviews, excluding one participant that exited the organization due to an end of contract. Appendix C.1 describes the full set of interview questions used in the member-checking interviews.

### 3.5.2 Coding Process

The coding process for our member checking interviews was identical to the coding process in the first phase of interviews. Appendix C.2 describes our full codebook for the member checking interviews.

Similarly, during the data analysis, two independent researchers coded and compared notes to reach a substantial inter-rater reliability >0.60, using the Cohen Kappa's coefficient for measuring observer agreement for categorical data [58].

# Chapter 4

# Findings

This chapter presents the practices and limtations for building a shared understanding of NFRs, and the practices for improving remote collaboration when building a shared understanding of NFRs. These findings are supported by participant observations and interviews with the software development team at Alpha.

## 4.1 RQ1: How does a remote software organization that adopts CSE practices reach a shared understanding of NFRs?

First, to understand how the software development team views NFRs, we asked participants to tell us which NFRs were vital to the success of the projects that they work on. In our data analysis, we recognized three major NFRs that were mentioned repeatedly: USABILITY, PERFORMANCE and MAINTAINABILITY. We also recognized a pattern where the NFRs mentioned during the interview sessions differed depending on the role and level of experience of the participant. For instance, we observed that most of the junior developers with less than 2 years of experience were more familiar with USABILITY than NFRs like SECURITY or TESTABILITY , as most of Alpha's products included a lot of visual components. Additionally, front-end developers and designers mainly described USABILITY when referring to NFRs e.g.

> "Well, usually the biggest NFRs that we get are everything to do with how it looks. When you're dealing with visual specs, the most important thing is usually how it behaves in different environments".

Similarly, we observed that the engineering leads and managers made reference to other NFRs such as PERFORMANCE and UNDERSTANDABILITY e.g.

> "But you could definitely consider there to be NFRs for our public APIs. And the things that you want to consider in those cases are, does the API make sense? Is it familiar for other developers? Is it something that they could easily discover?".

At Alpha, the managers and engineering leads have more experience within and outside the organization which leads to more knowledge of the NFRs that are vital to the success of their software projects. In another instance, a manager describes SECURITY as an important NFR for their clients. e.g.

> "They want to know that the data that they are providing us ... is secure, it's deleted when they need it to be deleted, and that the transmission is secure as well, that it's encrypted, and that there is no way to intercept the data as they're sending it to us or retrieving it".

We observed that the managers and engineering leads provided more details about how NFRs are managed within Alpha. For instance, a manager at Alpha described how they monitor the PERFORMANCE of their applications using an application performance monitoring tool, DataDog [23], for visualizing several performance metrics such as the response rate of API calls. The manager also acknowledges the need for more efforts around real-time performance monitoring of their software applications to enable faster decision-making around resolving issues or errors.

We then asked participants how they reach an understanding of NFRs for any project they have worked on. From our data analysis, we identified five practices at Alpha, outlined in Table 4.1. In addition to the codes from which we developed these themes, Table 4.1 also provides information as to whether our data indicates that the shared understanding was *implicit* or *explicit* [40], whether shared understanding was the result of *triggers*, as found in the work of Werner and colleagues [87] and correspondingly whether shared understanding was *proactive* rather than *reactive*.

| **Practices** | **Description / Key Phrases** | **Implicit / Explicit** [40] | **Triggers** [87] | **Proactive / Reactive** |
|---|---|---|---|---|
| **Validating NFRs through feedback** | Validating NFRs through code review<br>Eliciting NFRs during feedback<br>Capturing NFRs from demos | Implicit | Rework | Proactive & Reactive |
| **Deepening the understanding of NFRs through experience** | Learning how the organization values NFRs<br>Learning about NFRs over time<br>Limitations due to lack of experience | Implicit/Explicit | Change in scope<br>Rework | Reactive |
| **Wireframing interface designs** | Sketching solutions<br>Designing solutions using visual aids | Explicit | N/A | Proactive |
| **Discussing problems and solutions** | Meetings for implementing NFRs<br>Informal conversations about problems | Implicit | Change in Scope<br>New requirements | Proactive & Reactive |
| **Asking questions** | Asking questions about NFRs<br>Asking questions formally or informally | Implicit | Change in scope<br>New requirements | Proactive & Reactive |

Table 4.1: Summary of practices for building a shared understanding of NFRs (RQ1)

The five practices for building a shared understanding of NFRs as observed at Alpha are enumerated and described below:

1. Validating NFRs through feedback
2. Deepening the understanding of NFRs through experience
3. Wireframing interface designs
4. Discussing problems and solutions
5. Asking questions

### 4.1.1 Validating NFRs through feedback

At Alpha, developers and managers provide and receive feedback from stakeholders during the software development process, as enabled by their use of standardized communication tools such as Slack and Gather. Alpha benefits from the quick feedback loops of CSE through daily meetings, other regularly scheduled or unscheduled meetings and informal conversations that happen during the software development process. These meetings mostly happen through Gather, the virtual office, e.g A developer describes how Gather is used for daily meetings.

> "We Slack or meet in Gather with screen-share and also every day, we have a morning meeting where we check-in and show what we're working on. And there can be opportunities for feedback there as well".

At Alpha, NFRs are sometimes validated during these meetings, for instance, a manager describes a scenario where during a demo meeting, PERFORMANCE was captured as a valuable NFR. e.g.

> "I think we've actually seen this quite recently, with the demo project that's been happening here, where the engineering team was tasked with actually implementing the program. And by simply implementing it, you run up against all these NFRs that should exist. Because now you're going like, hey, I went to use it, it was really hard, I went to use it, it's taking forever to load".

In this instance, the manager acknowledges that the demo with the stakeholders provided a means for the developers to validate the performance requirement. In

another instance, a software developer describes the project meeting that helps with discovering NFRs like EXTENSIBILITY e.g

> "Because when the shaping is happening, we're starting to think about those NFRs about how it might work for other use cases".

In this instance, the "shaping" meeting refers to early meetings with the stakeholders where the team attempts to define the initial architecture and requirements for a new project. We observed that the output of these meetings were usually early versions of assets such as data flow diagrams, user flows, user stories, API schemas or program designs created and shared through Figma or Coda [19]. The early versions of assets created are then used to refine the software requirements during project kick-off meetings with the developers and project engineer. In addition, the project engineer uses the early version of assets to create more detailed executable specifications, and define use cases on stencilbooks or storybooks. Similarly, our participants express that there are similar meetings such as daily download meetings, code review and deep-dive meetings, that encourage the validation and capture of some NFRs i.e, TESTABILITY, MAINTAINABILITY. e.g.

> "Whereas the NFRs happen more passively. They're things that are built into processes, for example, when you create a pull request, there's a template that you have to fill out when you do a product launch ... And then obviously, there's things like code review, review and feedback from the managers that sort of catches the NFRs".

In addition, a developer describes these channels for feedback as a way to actively validate NFRs like TESTABILITY. e.g

> "... And so if things are getting missed, like, tests or specs, that's an opportunity to get feedback, but really, it's any channel of feedback. So that could be slack messages, code review or just conversations throughout the day on Gather".

A manager further describes a scenario where code review revealed that the TESTABILITY requirements were not met. e.g

> "... I was doing some code review for a refactor inside of a existing component ... So the code was written in a way where it couldn't really be unit tested. It was hard to happen as a state was leaking in and it was making it very difficult to unit test

> ... So that was just about the way that we wrote our views. We needed to change the way that we wrote our views in order to do that".

In this instance, the manager acknowledges the need to improve how some of the design components were written as this affected the ability to effectively test those components.

Furthermore, we observed that developers validate NFRs also during the Quality Assurance (QA) process. The QA process reveals bugs, new tasks or improvements that encompass both functional requirements (FRs) and NFRs. During one project, we observed that a developer flagged error handling issues during the QA process. The errors affected the USABILITY by developers that make API calls to the application, the SECURITY, and the RELIABILITY of the application. The errors were described in details and assigned to a developer working on the project through Coda. Usually, the developer that discovers the issue is not the same developer that resolves the issue; however, the developer that discovers the issue also verifies that it has been resolved at a later time (sometimes this task is solely assigned to the project engineer).

Using executable specifications with Cucumber enabled the software development team to perform end-to-end testing of their programs. At Alpha, we observed developers perform end-to-end program testing to verify that the specifications are complete and users can perform the actions described in the specifications. Usually, these tests can be done by the developers or in collaboration with other stakeholders; developers can raise, document or address any bugs or improvements discovered. Sometimes, when stakeholders suggest improvements, the software team does not implement them immediately; however, the team considers them and may implement them later as part of a much larger project. For instance, during one project, a customer suggested a specific change to the design of their program to limit the amount of data the end-users can see at a time. This change affects the USABILITY of the application by its end-users. The software development team documented this feedback but did not include this improvement with the customer's product at that time. Instead, it was added as a change in the standard design components that can be reused and extended for other applications.

### 4.1.2 Wireframing interface designs

We observed another way that Alpha reaches a shared understanding of NFRs is through wireframing and utilizing visualization tools. Our participants express that they mostly used Figma, sketches or minimal designs to describe some of the NFRs during project meetings or informal conversations. e.g. A developer describes how Figma is used regularly by the team.

> "And that usually involves using something like Figma, or Fig Jam to quickly visualize what we're talking about. So really basic forms, like shapes, and just text, and blocks of lines that connect different screens".

To mitigate the challenges that could occur when working remotely with visual components, developers describe the use of screen-share on Gather, as a way to make it easier to reach a shared understanding of NFRs. e.g

> "A kind of advantage of Gather is that you get to share your screen and [person] likes to write diagrams and stuff like that or just mock-up code on the screen, which definitely makes things easier to understand".

In this instance, we observed that when developers share their screen to describe or mock-up code, it also helps with solidifying their own understanding of the NFRs. For instance, concerning TESTABILITY, we observed that when a developer shares mock-up code for test cases, the developer often talks about the test coverage and reasons for the test cases. The developer also receives feedback from senior developers or managers on the ways to improve the test cases or better ways to approach testing for that application.

In addition, a manager further describes the effectiveness of wireframing interface designs and utilizing visualization tools by stating

> "I think that's where things like shaping, wire-frames, starting to try to think through that really helps. Design drawings, I find are really helpful. Early mock-ups help people because I find that a lot of the team does well with actually experiencing something. They quickly start to see ... this does the thing, but it's totally unusable for other reasons".

For NFRs such as USABILITY, the software development team at Alpha visualizes what needs to be built and requests feedback on those designs from other developers

and clients. e.g. A manager states

> "If we take the NFRs that are typically involved in something like visual design, in terms of user experience, obviously we have our designer who will design something, but we do try to obtain as much feedback on that design as possible. If possible, we actually like to communicate that or publicize it with the client, if there is something that's client-specific".

In addition, we observed that the software development team at Alpha often used mock-ups, demos and wireframes to elicit feedback from stakeholders, especially customers, on NFRs like USABILITY. For example, a manager explains this as

> "When we're talking about user experiences, they start as very rough wireframes ... And I think, showing it off early and often is vital for creating alignment on how it needs to behave".

We observed that when developers visualize some of the NFRs for a project through Figma, it serves as a common reference for the developers involved in that project. For instance, we observed that the software development team at Alpha uses a technique called whiteboarding through Figma to discuss aspects of the project and document decisions around some NFRs such as the USABILITY, SECURITY or PERFORMANCE. A developer describes whiteboarding as

> "Oh, there's a lot of times where [person] will be drawing a diagram. And then you can also just drop in a block of code ... And then you can just like drop a fully functional code editor, inside of the diagram, basically, it's just better".

Furthermore, we observed that the managers often have formal or informal meetings with the junior developers, where they use visual descriptions through Figma to explain core concepts related to the ongoing project or recent projects. For example, a junior developer explains this for USABILITY requirements,

> "... when I'm talking to [person], he almost always explains things to me while sketching something out. It is very helpful to visualise exactly what he's saying, even if what he's sketching isn't how it's actually supposed to look, but more like a diagram of ... how the user should go through and interact with it".

### 4.1.3 Deepening the understanding of NFRs through experience

Four of our participants expressed to us that they deepen their understanding of NFRs over time through the projects that they work on within Alpha. These NFRs are sometimes not explicitly defined but incorporated into already existing processes and the developer learns how to implement them through those processes over a period of time. e.g A developer states

> "There's just a process that isn't formally presented, it's just picked up over time through training. I think that the majority of NFRs ... have been picked up that way".

In other instances, developers told us that a helpful way to deepen their understanding of NFRs is through learning how NFRs are handled in previous similar projects. e.g. A developer describes how previous projects have helped with deepening their understanding of NFRs

> "I think the biggest tool that has helped me learn NFRs has really just been reviewing previous work, reviewing previous specs, reviewing previous integrations, things like that, because it's the best way to get a sense of the expected code quality and expected way of doing things".

In this instance, the developer learns more about code quality, related to code hygiene and the readability of code, which affects both the USABILITY of other developers and the ease of MAINTAINABILITY of the code by other developers.

Furthermore, a developer describes how Alpha keeps people connected through Gather while working remotely and how that also helps with deepening the understanding of NFRs e.g

> "I did mention earlier that Alpha is remote first but I do find that they do a good job of keeping people connected. And I think Gather town has really helped with that, it's just a place where you can feel like you're next to the people that you're working with. And they're, a few button clicks away from a call to talk about things and screenshare".

At Alpha, developers document software requirements as executable specifications using a tool, Cucumber [21] that supports Behaviour-Driven Development (BDD).

Developers at Alpha describe this process as "specification writing" and acknowledge that there are requirements for how they write these specifications. Our participants describe specification writing as an important NFR because it influences the degree to which an application supports testing (i.e user testing) and how application tests are written or understood. For example, a developer explains

> "... most of the testing we do accompanies Cucumber specs. So Cucumber specs are at the base of most of our automated tests".

However, we observed that some of our participants expressed some difficulties in understanding specification writing due to a lack of experience. e.g, A developer states

> "I still think I'm learning what the requirements are for spec writing, because it's just it's a nuanced aspect of the requirements ... So it's something that I'm still learning, but definitely have a better understanding of than when I first started".

In addition, a software developer describes learning Alpha values as an important factor to building a shared understanding of NFRs e.g.

> "I guess the tricky thing is that a lot of getting shared understanding, I think is almost kind of unwritten, it's like, you have to get a sense for how your organization values those NFRs to be able to kind of work with others around you".

In this instance, the developer acknowledges that the culture at the organization affects how the software development team collaborates and how NFRs are prioritized and managed.

### 4.1.4 Discussing problems and solutions

Developers express that they sometimes gain more understanding about NFRs through discussions during regularly scheduled meetings on Gather, such as *bi-weekly guild meetings*. e.g A developer describes guild meetings as

> "For the [name] guild, we have bi-weekly meetings ... And that's to go over emerging standards or libraries that we want to adopt. And also [person] will sometimes organise a coding exercise, like 20 minutes, then we compare results and learn from there".

In this instance, the developer describes guild meetings as a way to discuss and share standards for some of their NFRs and how they learn through coding exercises organized by managers or senior developers. When we spoke to a manager on the software development team, we learned that guilds were created as a way to centralise the people that work on a specific aspect of projects. For instance, developers actively involved in the design and USABILITY of projects would belong to the same guild. Therefore, guilds are mainly skill-focused and the goal is for developers to help each other learn and improve the specific skills related to their work areas. A developer explains how guilds helped improve their understanding of TESTABILITY as

> "... and writing tests, so I talked with other members in the [name] guild. So that also helps me learn how to actually write tests, how to write good tests, how to test certain things that I've found, that I did not know before, like testing a function that takes in a function as a parameter and returns another function, things like that".

In addition, participants also describe their communication through Gather, as being key to facilitating several feedback cycles and meetings where it feels like they are working face-to-face. e.g. A manager states

> "I can think of one software that we use, which is Gather. And Gather is really obviously a video communication tool. I think it does foster the ability for individuals to kind of interact more. And by interacting more, and by kind of stopping by somebody's desk that you may not have talked to in a while, you can kind of ensure that communication of different departments is kind of happening, and it's happening more organically".

### 4.1.5 Asking Questions

Through several feedback methods, we observed participants ask questions to reach a shared understanding of NFRs. A manager describes asking questions to clients to elicit the NFRs that are important to them during the early stages of discussing the project. e.g.

> "... you want to ask a lot of questions, you want to poke a lot of holes and things and, really go to the very edge of, you know, when you're sketching it out. So yeah, I would say that initially, it's a lot of questions".

In a separate instance, a developer uses question asking to clarify an NFR and ensure that all members of the team have a good understanding of what the goal is. e.g Concerning usability,

> "And you have to ask questions and make sure that the person fully understands what the goal is. Sometimes things can be unclear, but so long as you're constantly asking questions, then the person is confident that they know what they're doing".

Furthermore, we asked participants about what methods of communication have been most effective for reaching a shared understanding of NFRs. All of our participants mentioned Gather e.g, A developer says

> "And then we also have a virtual office environment called Gather, which is what we mainly use. I think Gather is good because it's more multimedia because you can talk and have video and share screens".

Finally, we asked participants when they know team members, inclusive of themselves, have a common or shared understanding of the NFRs. Developers describe this as when there is minimal feedback or questions during project meetings or conversations. e.g. A developer describes

> "Like when we go into the 'deep-dive' we know when the non-functional requirements have been met, when we as a group meet, and there aren't gaps in our shared understanding. We come out of a review meeting and there's not a lot of feedback".

A manager describes this as when the team members are able to discuss the problems and solutions with each other as well as describe the behavior of the software product e.g.

> "I'm kind of sufficiently satisfied with two conditions ... which is when the team members are starting to be able to explain the problem and understand the detailed nuance explain that problem to others. The other one is when you start to see like acceptance criteria that's really starting to look like one of our actual user acceptance tests. And by that point, you get a sense of if someone really understood the non-functional".

However, our participants acknowledge that shared understanding of NFRs should continue to evolve and there may not be a way to guarantee that there is a complete shared understanding of NFRs within the team at one time.

A manager expresses this by saying

"I don't actually think you can have guarantee, honestly, until it's actually in a client's hands being used or users hands. Because, I mean, there's a chance we got the NFRs wrong, you know, every single person did".

In another instance, a manager states that

"I don't think it's possible to reach 100% shared understanding, there's always going to be areas where people are running on assumptions, or people might have different interpretations".

## 4.2 RQ2: What are the impediments to the shared understanding of NFRs in a remote software organization that adopts CSE practices?

We asked participants if they experienced any challenges with understanding NFRs when working collaboratively. From our data analysis, we found five main themes that describe the impediments to building a shared understanding of NFRs:

1. Gaps in knowledge
2. Gaps in communication
3. Limited understanding of customer context
4. Individual differences
5. Unspecified NFRs

Additionally, we identified the main triggers across these impediments. The themes, triggers, and associated codes are in Table 4.2.

| Themes | Sample Codes/Key Phrases | Triggers |
|---|---|---|
| **Gaps in Knowledge** | Insufficient background knowledge of a project<br>Lack of experience with technical requirements | Rework<br>Change in scope |
| **Gaps in Communication** | Making assumptions about a project/process<br>Unclear communication of expectations | Redesign/Reshape of projects<br>Scope creep<br>Change in scope |
| **Limited understanding of customer context** | Learning about customers<br>Insufficient understanding of customer needs | Scope Creep |
| **Individual differences** | Differences in ideas and background<br>Differences in skills and knowledge<br>Differences in approach to work | Rework |
| **Unspecified NFRs** | Uncommunicated or unclear NFRs<br>Unwritten NFRs<br>Unclear NFRs | Difficulty implementing NFRs<br>New requirements |

Table 4.2: Summary of impediments to building a shared understanding of NFRs (RQ2)

### 4.2.1 Gaps in knowledge

We observed that a majority of the software developers at Alpha have less than three years of experience, and this constitutes a difference in the way that they gain, understand and share knowledge. During daily work activities, we observed that these developers often met with senior developers and managers for direction and to validate their work. In several instances, we observed that the managers organized several informal meetings with the developers where they sometimes pair-program with them or explained technical requirements and some of the development standards that should be followed.

We observed that, in some instances, developers experience a gap in knowledge of technical requirements where they seek clarifications from other developers or managers working on the project. A manager at Alpha describes this as

> "... So somebody might feel like the communication is unclear because they don't have the knowledge required to have the understanding ... And so, when I'm thinking about shared understanding, a lot of it starts with what knowledge people have ... the biggest misunderstandings around NFRs typically come from a lack of knowledge".

In other instances, our participants describe a gap in knowledge as when they have limited knowledge about a project or changes to a project. This often leads to triggers such as *rework* and a *change in scope*. Concerning USABILITY, a developer narrates a scenario where *rework* occurred due to a gap in knowledge about the requirements for a design component.

> "... I don't think I had enough background knowledge on exactly what the requirements were, the use cases were for that tag or the packages. And because it was a small part of a very large project, there wasn't any sort of, like kickoff, or anything or any sort of like meeting to talk about that small aspect of it".

In this instance, the developer understands that the rework occurred because they did not have sufficient background knowledge about the usability requirements. The developer also acknowledges that this may have happened due to the requirement being a small aspect of a much larger project.

In a different instance, a manager expresses the challenge with sharing knowledge as well as the need to share knowledge within the team more often, e.g,

> "... [person] and I have a lot of knowledge, having built these programmes over the past three or four years, and that is beneficial as a one-person team. But as the team builds out, the challenge is getting that knowledge in somewhere, having it be accessible".

### 4.2.2 Gaps in communication

In several instances, our participants describe gaps in communication within the software team and gaps in communication between the software team and stakeholders due to unclear expectations for projects or assumptions about projects, which may lead to some rework or a change in the scope of a project. A manager describes a scenario in which the software team made inaccurate assumptions about what the stakeholders expected for a project regarding USABILITY, which led to discussing additional requirements that were not within the scope of the project,

> "They had made the assumption that they would be able to go in and update these prints after launch. And we had made the interpretation that everything was just sort of hard-coded, write once and forget it. And so, in the end, it was something that we actually had to push back on the client and say that, making that something that's editable, would significantly increase the scope".

In this instance, the manager acknowledges the gap in communication between the software team and customers led to a *scope creep* where the clients requested requirements beyond the scope of the project.

Another member of the software team describes a situation where there was some unclear communication regarding the project expectations for USABILITY requirements which resulted in the developer working with inaccurate specification and eventually *rework*. A developer states

> "There have been times where I've had a conversation with [person] about what a product needs to do and how he has envisioned it. And I guess some things just maybe got lost in translation. And I end up working on something that is different or works differently than what he had imagined. And vice versa".

### 4.2.3 Individual differences

Differences in the background, skills, experience and ideas of team members can sometimes lead to misunderstandings of NFRs, especially when there is no common foundation of knowledge. Alpha recognizes that having a diverse software team means that team members would have a different level of knowledge, skills and experience. e.g A manager states

> "the first thing that comes to mind when working with other team members, especially in a leadership position, is understanding that not every team member has the same level of technical or work experience as any other team member".

However, sometimes team members still make assumptions about how other members of the team think and how much they know and this may lead to a disparity in ideas about the product and how to implement the NFRs of the product. e.g. Concerning EXTENSIBILITY, a developer explains

> "But it just trickled down to things like the language we're using, just wasn't aligned even, people didn't even have the same ideas about what the functionality was. And that just like continued to trickle through the other non-functional requirements because it wasn't meeting some of the use cases anymore".

In this instance, the developer acknowledges that there was a lot of disparity in the way that the team members understood the FRs and NFRs. This situation led to the *redesign of the project* and several meetings among the developers involved in the project to re-align their understanding of the project.

### 4.2.4 Limited understanding of customer context

A major challenge with reaching a shared understanding of NFRs when working with customers is the insufficient understanding of the customer's business, products or needs. The lack of contextual knowledge about the customers can sometimes lead to misunderstandings or assumptions concerning the NFRs that are important to the customers, especially when Alpha needs sufficient information about their customers or their customer's products to deliver a product successfully, as observed, a developer explains

> "if I'm unable to understand what the motivating use cases are if I'm unable to see how this project actually came to be, what the scope is even, I'm unable to go dive into what those NFRs are".

Similarly, a participant expressed having experienced difficulty when working on a project and not understanding EXTENSIBILITY as an NFR. This difficulty was mainly due to a limited understanding of the customer and their products. e.g. A developer states

> "I think a lot of the confusion probably also came from it being my first couple projects here. And so I was still wrapping my head around what our customers actually are, how we interact with them, the full range of our products, and how they're planning on implementing their programs".

In this instance, the developer acknowledges that they did not fully understand the customers and product, which led to some *rework*.

### 4.2.5 Unspecified NFRs

We observed that some of our participants experienced difficulties with understanding or implementing NFRs due to unclear or unspoken expectations around these NFRs. Concerning MAINTAINABILITY, how a code-base can be understood and built upon with well-written documentation, a software developer describes difficulties. e.g

> "Definitely, for the specification writing for that project, it was pretty difficult for me, because I wasn't really sure what they should look like. And also there weren't great examples that I could base it off".

When NFRs are not clearly defined, communicated or documented, developers are often left with their assumptions on what NFRs to include for a project and how to implement the NFRs. e.g a developer explains

> "The thing is non-functional requirements are not always communicated as much ... trying to figure out what or not, understanding or communicating expectations around some of those NFRs from stakeholders".

Our participants expressed that this could happen, however, due to the frequent feedback loops within the software team, there are several opportunities to communicate and re-elicit the NFRs relevant to the work being done. e.g

> "The good thing about developing at Alpha is that our loops are pretty quick, we have a lot of communication, so becomes quite clear when something isn't being met. And when that comes up, just work to try and elicit what those NFRs are".

## 4.3 Practices for improving remote collaboration when building a shared understanding of NFRs

To validate our findings from the participant observations and the first interviews, we conducted member checking validation interviews where we asked participants questions about our findings to access how much they agree or disagree with our results.

Member checking is a useful process for validating the results from a qualitative research study [74] as member checking can uncover additional results or strengthen the research evidence.

From our data analysis of the member checking interviews, we recognized three additional themes that add to our understanding of how Alpha builds a shared understanding of NFRs, with focus on the practices for improving remote collaboration. We explain these practices below:

1. Building Connectedness through Gather
2. Sharing development standards
3. Cross-functional communication through guilds

Additionally, we found themes that represent the practices for building a shared understanding of NFRs (from our first interview data analysis). In summary, the themes from the member checking data analysis show that our participants agree with our findings. The codebook from the data analysis of the member checking interviews is described in Appendix C.2.

### 4.3.1 Building Connectedness through Gather

The software development team at Alpha is remote-first and so collaboration and communication is fully remote through Gather [35]. Gather is a virtual video conferencing tool that tries to simulate the face-to-face physical workspace. As remote

collaboration comes with several challenges such as limited social interactions, our participants describe the importance of Gather to their daily activities with emphasis on how Gather helps the team stay connected while working remotely. A manager explains that Gather makes it easier to have conversations with other developers because developers can see each other's avatars to know when they are available. i.e

> "... it makes it quite clear for who is available and who is not available to participate in those conversations. And it makes it easier for a conversation to expand".

Developers also describe Gather as being easier for quick informal conversations or "small talk" as it removes the additional effort needed to create and share a meeting link or wait for a meeting link to be accepted. e.g

> "I feel like with Gather, it's a lot easier to quickly talk to someone. Whereas Zoom, that's usually planned and scheduled. And there's a link that has to be sent over and whatnot. And like, it's like all the little things that kind of add up to making someone not really want to reach out and like schedule a call?"

In addition, many of the developers and managers rely on Gather for social interactions with their colleagues, a manager describes this as

> "I would say more of a personal connection with people, you have to perform the physical act of like going to their desk and seeing them and like interacting with them. It also gives you kind of some social awareness".

In this instance, the manager describes the activity of "going to their desk" on Gather, enabled by Gather's user interface that allows team members to use avatars to 'walk' around an office space and interact with other team members. We observed that developers often casually walk to other developers to ask questions or talk about their day. Gather also allows a developer or manager to expand conversations to include more team members on the fly without additional planning or scheduling. A developer describes this by saying

> "... So if somebody that you need to talk to is talking to somebody else that you need to talk to, great time to go and kind of jump into the conversation. But if somebody is in kind of like a closed door meeting, well, you kind of have a better sense that it is not the appropriate time".

Furthermore, the managers at Alpha explained why the organization switched to using Gather for remote collaboration. First, the managers describe Gather as a

tool for fostering a sense of community within the organization. Secondly, a manager explained that Gather helps improve their employee's sense of togetherness and awareness, positively affecting their mental health. At Alpha, supporting mental health is essential and is a significant benefit that Gather provides.

### 4.3.2 Sharing development standards

We observed that our participants emphasized how shared development standards helps them understand NFRs and how to implement NFRs. A manager explains how the development standards helps with implementing non-functional requirements like USABILITY, described as user experience.

> "There's obviously some important standards that we push towards user experience ... Great user experience is one of those things that's difficult ... When it comes to user experience, we've centralised the people that make user experience decisions into a group and then we have a standards in that group".

For the managers, sharing development standards creates an alignment among the developers who work independently on different projects. e.g A manager describes development standards as

> "... what are some of our important design standards, form standards, writing standards, visual layout standards, as well as standards for feedback on designs to confirm that they're good, as well as more recently user testing".

For most developers, the development standards are a guide to understanding how the organization values NFRs and the expectations for several NFRs. For MAINTAINABILITY, specifically specification writing, a developer describes a few of the shared development standards for how to write good specifications as

> "I noticed basic ones, just always writing in active format, instead of passive and then writing in current, present tense".

At Alpha, developers and managers share development standards through guilds, a manager explains

> "So when we have a guild meeting, for instance, we'll talk about what we've been discovering, as we've been doing a role, how that fits into our wider set of standards".

In addition, guild leaders ensure that the development standards are well documented and communicated across the software development team and organization. The guild leaders are also responsible for improving how the developers understand the development standards through organizing coding or design exercises periodically e.g. For MAINTAINABILITY, specifically specification writing, a guild leader explains

> "But then also, just last week, we ran a spec exercise, where we took an existing piece of functionality in existing specs that were written, I think, like five years ago, right far beyond far before we had established standards. And we looked at it and we reworked it".

Similarly, for NFRs like USABILITY, developers share and learn about the development standards through the guild meetings. e.g

> "... So a lot of times our conversations in the [name] Guild are about setting standards and like how we should set these processes and whatnot".

Developers describe how the software team sets some of the development standards informally during project discussions and then they may be modified and explained in greater detail at a later time.

> "Then [person] came over and pulled out a virtual drawing board and we documented, okay for in this case, in these cases, the forms will look like this. These are the standards that we are going to use, like going forward with forms".

In addition, developers explain that the development standards are sometimes created or evaluated during the code review by managers. e.g., for MAINTAINABILITY, specific to code hygiene or code quality, a developer explains

> "... [person] mentioned that when the devs have a code review, that kind of reviews sets the standards for how the code should be written and whatnot".

### 4.3.3 Cross-functional communication through guilds

We observed that the guilds at Alpha facilitate collaboration across various roles, teams and projects, for instance, often times a guild will consist of developers, designers, project managers, customer success representatives and sales representatives, etc. Therefore, the guilds enable cross-functional communication across the organization which makes it easier for stakeholders to collaborate or communicate throughout

the duration of a project i.e.

> "So that's kind of one and I think that cross-functional layer of communication, that goes right from the customer, to marketing, honestly, and back to the customer, again, can provide like subtle notes quite often in there".

In another instance, a developer describes how guilds enable cross-functional communication across teams as:

> "And there is improvements to be made in a specific role, where people also work on different products, like the admin portal, or the back end, or the integration. So guilds are this way to say, Okay, we're all doing the same job, even though we're not doing it in the same teams. But we should get together and make sure that we're doing this job to the best of our ability".

We observed that cross-functional communication across various roles and teams was necessary for the release management process. For example, Alpha introduced a guild specific to release management, and a crucial part of creating and sharing development standards for this guild was introducing specialized training. These specialized training often included various teams and roles at Alpha and so served as a way to further solidify the necessary cross-functional communication among team members. A manager explains how members of the guild collaborate as

> "So the release managers would work with other departments and other people in the product team to create that documentation that can be shared externally with clients".

At Alpha, guilds enable frequent cross-functional communication across various teams so that project stakeholders can regularly coordinate their activities, which is essential in remote-first organizations.

# Chapter 5

# Discussion

CSE advocates for ongoing customer feedback and rapid iteration cycles [33] [40]. Our study focuses on specific continuous activities in CSE, namely continuous verification, continuous testing, continuous integration, continuous deployment and continuous delivery. Our work focuses on these main continuous activities because they are common and predominant in prior research studies and they relate directly to our study of NFRs. The iterative approaches in CSE such as continuous verification and continuous testing, leads to earlier verification of requirements so that errors can be identified and fixed quickly while the context is fresh in the developers' minds [33]. However, prior research indicates that CSE may pose a detrimental effect on the shared understanding of non-functional requirements [87]. In remote organizations, building a shared understanding becomes more challenging due to the additional recognized barriers with remote work, such as challenges with information sharing and team members having a reduced ability to build trustworthy relationships [56].

For an NFR like TESTABILITY, our study focuses on both the testing of requirements, specifically NFRs, and measure of the ease of testing a piece of code or functionality. Our work explores the delicate balance of achieving a shared understanding of NFRs in a remote CSE organization (Alpha), and in this section, we discuss the findings of our empirical study.

Through our research, we uncovered five practices from Alpha to build a shared understanding of NFRs: validating NFRs through feedback, deepening the understanding of NFRs through experience, wireframing interface designs, discussing problems and solutions, and asking questions.

In addition, at Alpha, we found some impediments to building a shared understanding of NFRs. For instance, the software team experiences gaps in communication

that may occur when team members make assumptions about each other's knowledge. In addition, sometimes, developers have a limited understanding of customer context, which may occur due to insufficient elicitation of NFRs or the limited knowledge of the customers. We discuss how Alpha mitigates challenges with building a shared understanding in a remote setting through the use of multiple collaboration tools, particularly Gather.

We believe these findings have important research implications concerning the intersection of shared understanding of NFRs and CSE in remote collaboration. For instance: how an organization could use the practices observed at Alpha, in particular, practices that are proactive, rather than reactive, to build a shared understanding of NFRs.

## 5.1 Shared Understanding of NFRs in CSE

Glinz describes implicit shared understanding as the mutual understanding of non-specified facts or assumptions [40]. Our study suggests that Alpha relies on *implicit* shared understanding of NFRs, partly due to CSE practices that encourage frequent feedback [33] through formal, informal or face-to-face communication [25] [27], as observed

> "a lot of getting to shared understanding is almost kind of unwritten".

Our findings corroborate and bring additional, empirical detail about the amount of implicit shared understanding, as four of the five practices used to develop a shared understanding of NFRs were in fact implicit, namely validating NFRs through feedback, deepening the understanding of NFRs through experience, discussing problems and solutions, and asking questions. Only a single practice, wireframing interface designs, was explicit.

At Alpha, one of the practices for building a shared understanding of NFRs is by validating NFRs through feedback. Our results show that developers at Alpha validate NFRs mainly through tests and manual validation through code reviews or *deep-dives* or daily meetings. Concerning some NFRs, such as TESTABILITY, Alpha uses automated test coverage tools to regularly ensure that the code-base meets all of the pre-defined testing requirements. However, for other NFRs, such as USABILITY, Alpha relies heavily on manual verification and validation through user tests and by MEASURING EMOTIONAL RESPONSES.

Some NFRs, such as USABILITY, are not easy to validate through automated tests

[89][88], especially in CSE environments. At Alpha, developers try to mitigate this challenge by defining and sharing development standards that describe how some NFRs are managed e.g For CONFIGURABILITY, a manager explains

> "... so I introduced a number of standards and built some packages to try and enforce those standards. So there's the standard for how we, on the server-side, get configuration from the environment".

In addition, a CSE organization like Alpha benefits from capturing NFRs directly in source control [88] by using code and related artifacts to define rules that represent the metrics for an NFR. Werner et al. describe capturing NFRs directly in source control as setting an objective metric for developers who may not have enough time to acquire the knowledge of what constitutes an acceptable NFR threshold [88]. At Alpha, the software team uses automated dashboards on DataDog [23] for monitoring PERFORMANCE metrics such as error rates or API response rates. The explicit rules that represent the performance metrics are in source-control in the form of version-controlled JSON configuration files. The process of putting an NFR directly in source control helps with sharing knowledge about what the organization defines as an acceptable threshold for the NFR, leading to an increase in shared understanding of the NFR.

Werner et al. describe the importance of the shared understanding of configuration management or configurability. Configurability is an NFR that encompasses process quality [88], and supports the rapid development, configuration and deployment of software products. Prior research has described comprehensive configuration management [49] for continuous integration and continuous delivery as a key component of CSE. Through continuous delivery, the software team at Alpha can deliver new features, configuration changes or bug fixes to the end-users safely and quickly [5]. Many organizations use more than a single configuration tool, such as Docker, to create a software environment and applications, as observed by Werner et al. [88] within the three organizations in their study. At Alpha, developers use an in-house configuration tool for setting up their environment and application. Developers also use Process Street [71], Coda, and GitHub for documenting configuration standards and collaboration, as these tools allow the developers to coordinate their activities around configurability. An organization would benefit from continuous delivery incorporated in CSE practices when they invest in building a shared understanding of NFRs such as CONFIGURABILITY.

At Alpha, the software team benefits from CSE continuous activities such as continuous verification through the practice, validating NFRs through feedback for building a shared understanding of NFRs. For example, at Alpha, the software team continuously verifies MAINTAINABILITY with the project stakeholders at various stages of the software development process through *deep-dives*, demos, QA, and regularly scheduled weekly/bi-weekly meetings. These continuous verification activities allow the software team to align their understanding of the MAINTAINABILITY requirements and set or share the development standards for MAINTAINABILITY.

Furthermore, continuous integration allows the software team at Alpha to regularly automate their builds, test and validate their software products or services throughout the software development process. As continuous integration relies on testing and validating requirements, including NFRs, building a shared understanding of TESTABILITY is important for software teams to establish an end-to-end flow between customer demands and the fast delivery of the software product or service. When software developers build a solid shared understanding of TESTABILITY, they can be confident that software changes delivered likely do not introduce bugs or errors.

However, we hypothesize that CSE also hurts the shared understanding of NFRs at Alpha. Due to the emphasis on FRs and frequent software releases at Alpha, we observed that NFRs are sometimes unclear and unspecified. Due to the uncommunicated expectations for some NFRs, the NFRs are not documented formally or tested. A developer explains

> "I think we do a really good job of capturing functional requirements ... but we don't quite have the same sort of like formal process to grab non-functional requirements ... So, I think there probably could be a bit more of a process around kind of grabbing some of those early, for specific projects, I spoke about those like unwritten ones, which maybe should be documented".

In addition, when FRs are priortized over NFRs, NFRs are sometimes not documented or verified until a trigger occurs such as rework or new requirements beyond the scope of the project. A manager describes an experience

> "And one of their non-functional requirements is relating to not using HTML responses from APIs ... what ended up happening is, as we get into the implementation process, and we're doing data mapping, and kind of planning out how our different APIs are going to suffice their needs, it was brought up that it is one of their non

> functional requirements, which effectively meant that we need to go back to the drawing board on which API, we're going to provide them".

In conclusion, our findings suggest that organizations can benefit from CSE when building a shared understanding of NFRs, through continuous activities such as continuous verification or integration. However, CSE may hurt the shared understanding of NFRs when NFRs are de-prioritized due to the emphasis on frequent software releases, leading to triggers such as a change in scope or difficulties with implementing NFRs.

## 5.2 Mitigating the Effects of Remote Interaction on Shared Understanding

Previous research analyzing surveys during the COVID-19 pandemic has suggested that effective communication is essential for a software team's productivity. Communication is often affected by changes in a team's culture and inefficient communication could pose a challenge to reaching milestones [66]. As CSE organizations rely on frequent feedback cycles, the quality of communication within software teams may suffer when team members work from a distance, i.e. work remotely [26]. However, our findings suggest that Alpha was able to mitigate some of the challenges in remote communication by using Gather, a virtual video conferencing tool that tries to simulate the face-to-face physical workspace

> "Gather has made it possible for us as a remote team to have informal conversations. You literally just walk your little character up to someone else's little character, and you're talking to them".

To the best of our knowledge, there is limited research on Gather as a collaboration tool in software development and collaborative work. Our findings bring evidence about Gather's value in creating spaces that approach in-person interaction conducive to shared understanding in software engineering. A 2020 study of a distributed design team working during COVID-19 lock-downs [61] extensively used Zoom, Slack, and Google docs and identified challenges with remote collaboration. The design team reported difficulties establishing a team spirit in a distributed setting due to the limited one-view per participant experience with Zoom meetings during group sketching and the difficulties with understanding the team member's progress. At Alpha, Gather

allows the software team to achieve the practices in our study by mitigating some of the challenges of remote collaboration. For example, Gather simulates a co-located design space that encourages screen-sharing and automated video-conferencing, allowing regular and frequent collaboration within the software team. In addition, Gather allows the software team at Alpha to have informal communication through closed channels, simulating the traditional work environment.

A key enabler for shared understanding is the ability to ask questions and receive feedback, regardless of background experience, education, or other demographic factors [15]. We identified this as asking question in our practices for building a shared understanding. Previous research suggests other enablers of shared understanding in remote teams, such as visualizing information, online updates, and trustworthiness [56]. At Alpha, the software team visualizes information about the software product through the practice, wireframing interface designs, by using collaborative tools such as Figma and sharing screens through Gather. Additionally, the software team at Alpha stays informed about project changes or updates by discussing problems and solutions through regularly scheduled meetings or informal conversations on Gather.

We further discuss our insights about these enablers at Alpha. First, we observed that Alpha uses wireframing for interface designs and screen-sharing to enable the team to set and understand the expectations for NFRs. This observation also supports prior research on building shared understanding for supporting remote teams through building shared mental models with visual representations [41]. In addition, this observation supports prior research that suggests the existence of reference systems as an enabler for shared understanding [40].

Second, online updates like daily check-ins on Slack or formal or informal communication on Gather shows the importance of multiple forms of communication media [52] [44] and tools such as Gather, Slack, and file sharing. This observation supports previous research that suggests frequent communication ensures there is a shared understanding of how team members feel and how the project progresses [56]. Third, mutual trust enables team members to collaborate and share feedback openly, thus improving shared understanding within remote teams [56].

Previous research suggests that mutual trust is a prerequisite for implicit shared understanding [40]. Previous research also suggests that regular communication, including formal and informal communication, plays a vital role in creating relationships within remote teams and thus in building trust within the team, regardless of the difference in background, location or culture [43]. Alpha builds trust within the team

mainly through informal communication on Gather or Slack, where team members are open to seeking help from each other, as observed

> "If someone sends me a quick Slack message, and I look at it ... I will go straight to their desk and start to engage them in conversation so that we can whiteboard it or work through the problem".

Research evidence from our study suggests that developers at Alpha improve remote collaboration when building a shared understanding of NFRs through practices such as sharing development standards, building connectedness through gather and cross-functional communication through guilds. These practices allow the software team at Alpha to communicate frequently, align their understanding of NFRs through the shared development standards and build trust within the team. Our findings corroborates the recommendations by Werner et al [87] which suggests that shared understanding is enhanced through creating shared standards and effective communication.

## 5.3 Proactiveness in Building Shared Understanding of NFRs in CSE

Werner et al. [87] observed that an organization builds a shared understanding of NFRs primarily due to a response to accidental lack of shared understanding, related to various reasons – *triggers* – such as scope creep, rework, regulatory requirements, accumulating technical debt, needs of an important customer, or a disruption of service. In Table 4.1, we show the triggers that emerged from our data analysis and how they correspond to the identified practices. In the practices we observed at Alpha, the deepening the understanding of NFRs through experience is an example of such a reactive response, of where our data does not indicate that the response was the result of any of these or other triggers.

However, we categorized four of the five practices we identified at Alpha as being *proactive* in many instances, even though they were *reactive* in a few instances. Proactive practices do not occur as a result of a *trigger*, while reactive practices use different *triggers* to identify a need for building a shared understanding of NFRs [87].

One practice in particular sheds more light on how Alpha adapted to working and building shared understanding in remote settings: wireframing interface designs.

Through observations and the practices we identified in our study, we bring empirical evidence that further adds to and enhances the work by Werner et al.[87] on the *recommended* practices for building shared understanding i.e., how team communication and shared development standards form the fundamental base for building a shared understanding. Specifically, our study provides empirical evidence from actual observations of what Werner and colleagues' work posited as *recommended* practices for building shared understanding.

Our first practice, discussing problems and solutions, is part of using communication to develop a shared understanding. Developers can also initiate the creation of a shared understanding by asking questions, which is another form of communication.

Validating NFRs through feedback is a form of communication that can happen through continuous verification or testing or in a meeting; regardless of *how* that communication takes place, the emphasis is on the continuous validation of NFRs that can build a shared understanding. An organization can leverage wireframes for interface designs, a form of shared development standards, that may enhance the proliferation of a shared understanding. In addition, we have identified a new practice of building a shared understanding through experience; i.e., deepening the understanding of NFRs through experience. Through this practice, developers can learn how their organization values NFRs.

Prior research suggests that using virtual shared spaces that enable the team to visualize solutions and interact with visualizations encourages better communication and enables shared understanding [8]. Through Gather's virtual shared space and screen-sharing feature, Alpha used tools such as Figma, an online whiteboard for visual representations with elements like blocks and shapes, to support collaboration with team members, thereby enabling the team to develop a shared understanding quickly, as observed in prior research [8].

We observed that a *proactive* effort to build a shared understanding may not require a formal process or documentation. At Alpha, developers communicate and share development standards across several departments and teams through guilds, i.e.

> "There's a [guild_name] guild and we meet once every two weeks to sort of check-in on NFRs. It's what practices are we going to try and use? What's working, what's not working? It's been helpful for talking about how we build things and NFRs in specific areas of our jobs."

The loose agenda of these meetings is rather informal and lends to the creative discussions around NFRs.

Our findings indicate that tools such as Gather offer a medium for these informal, yet important, conversations to occur. Developers at Alpha unanimously describe Gather as a method that has created a sense of being connected to each other, despite their remote setting.

> "I think Gather town has really helped with that, it's just a place where you can feel like you're next to the people that you're working with".

Our findings have practical implications to software organizations as they show that while there is still a considerable amount of shared understanding built through reactive measures, there is also a sizable amount that is a direct result of some proactive practices.

One developer indicated that he found some NFRs as follows:

> "as you're going down into more details, during something like a journey map, that's where you start bumping into the non-functional requirements that may not have been clearly identified".

However, when it comes to such NFRs, discussions did not happen until during the software implementation, as one participant noted,

> "that's when you really start the back and forth communication on how you're going to address them ... So I think it's during that process of drilling down into the details on how a project is going to be run, that you start hitting them, and the NFRs".

Despite the complexity and importance of NFRs [85] [4], previous research observes that software developers discover NFRs during or after the software product implementation [65]. Eliciting NFRs at this stage may pose a challenge that could affect the entire software development process or lead to delays in software [65] [85]; therefore, it is prudent for an organization to ensure the conversations about NFRs are occurring early enough to facilitate building a shared understanding. The definition of *when* is left for future work; however, given the effects of architectural decisions on NFRs (or lack thereof), a shared understanding should be continually built throughout the development cycle, perhaps allotting enough time for "just-in-time" engineering [32].

# Chapter 6

# Threats to Validity

We acknowledge the limitations of our qualitative study through four components of the total quality framework [73]: credibility, analyzability, transparency and usefulness.

## 6.1 Credibility

For the *credibility*, the selection of Alpha may suffer from sampling bias as we focused on an organization willing to partner with us; however, we verified that their practices align with the continuous activities of CSE [33] through observation by the thesis author. In addition, we interviewed participants with various roles and levels of experience.

The interviewer-interviewee relationship is susceptible to bias that may have influenced the interviewee's responses to the researcher's questions due to the working relationship established during the period of observation. To limit the bias, we informed participants about our goal for the research and the importance of being transparent and honest with the responses to the interview questions. We described the purpose of our research to the participants, stating that our research study was to explore how the team a reached shared understanding of NFRs when working collaboratively. In addition, we performed member checking interviews to validate our research results. We also validated some of our research findings using the observation notes.

To ensure the safety and autonomy of our participants during data collection, we explained that study participation was completely anonymous and would not cause

any consequence or risk to the participants. We did this to create a comfortable space for the participants to feel relaxed during the interview sessions. We started each interview session by exploring the definition of shared understanding. Afterwards, we explained shared understanding in the context of our research with examples to ensure that each participant had the same level of knowledge about shared understanding. All of our participants provided details that were aligned with their role within the organization and level of experience.

The researcher's role in the study presents bias due to the researcher's preexisting knowledge of the subject area and prior industry experience with NFRs. While the researcher was familiar with NFRs and the broad concepts of shared understanding and CSE, the in-depth literature review for our study did not occur until after the initial data collection and analysis phases. The in-depth literature review was conducted after the initial data analysis to limit the influence of the literature review on the interpretation of the data.

In addition, for the thesis author, our research is a part of the requirements for completing their Master's degree, so the researcher has the typical motivation to graduate. While there is no unique way to mitigate this bias, we minimized the influence of the motivation to graduate by creating and validating a thorough research plan with our research supervisors and the HREB at the University of Victoria. Following our thorough research plan, we did our best to allow the participants to speak comfortably and allow the findings to emerge from the data.

Furthermore, the thesis author was embedded within the organization to ensure prolonged contact with the participants so that we can have a good understanding of shared understanding within the organizational context. We use a mixed-methods approach to our research by combining data collected during interview sessions with data noted during observations to confirm our results and build a solid picture.

Finally, we conducted member checking with our participants through interviews and presentation of our findings in two focus groups sessions to clarify our findings and strengthen our results. The member checking also revealed additional insights that add to our understanding of the research findings.

## 6.2 Analyzability

For *analyzability*, we used a transcription tool on the recorded audio from the interviews and one of the authors listened and verified each transcript. We performed the manual verification of each transcript to ensure that the transcripts matched the audio recordings. We used thematic analysis through the open, axial, and selective coding process from grounded analysis [47]; as described in Section 3.

We used the consensus process to establish inter-rater agreement [58] and align our codes and categories within our coding scheme. We conducted the consensus process with a second coder to unify our coding strategies and establish an inter-rater agreement of codes. During the consensus sessions, we recognized the differences in our interpretation of the data and discussed these differences and ideas.

To improve our interpretation of our research data, we used reflective memos during the coding process to describe patterns observed in the data collected for reference.

## 6.3 Transparency

For *transparency*, we provide a replication package (`https://doi.org/10.5281/zenodo.6273497`), and the appendices in this thesis which includes our codebooks, interview questions, and data about our inter-rater agreement sessions. However confidential data is excluded due to our non-disclosure agreement. In addition, we provide the recruitment documents used in our study and ethics approval document from the Human Research Ethics Board (HREB) of the University of Victoria.

Although we believe we reached theoretical saturation of codes in our data analysis, one limitation to our coding process may be the subjectivity of the exhaustiveness of our codes which may influence research results. We used tables to describe the main themes and observations from our research study.

## 6.4 Usefulness

For *usefulness*, we do not propose that our results and findings hold true for all remote CSE organizations. We acknowledge that our single case study research provides results limited to our partner organization. Hence, we provide insights into how shared understanding is built and practices for building a shared understanding of NFRs

relevant to requirements engineering and beneficial to similar software organizations such as remote-first CSE software organizations.

However, to increase the usefulness of our findings, more case studies like ours would be helpful with a focus on how remote software organizations can proactively build and maintain a shared understanding of NFRs, and how to evaluate a shared understanding of NFRs.

# Chapter 7

# Conclusion and Future Work

A shared understanding of NFRs is important for the success of many software projects. However, managing NFRs can pose a significant challenge to any software organization due to factors such as the conflicting nature of NFRs and the lack of shared understanding of NFRs. In addition, shared understanding of NFRs in CSE is still understudied and there is insufficient literature to show how organizations can effectively build a shared understanding.

While prior research has explored how NFRs are managed in CSE, considering the challenges and best practices for managing NFRs, there is still the question of *how* CSE organizations can build a shared understanding of NFRs. This thesis describes the practices for building a shared understanding of NFRs, through a case study of a remote CSE organization. Through discussing the practices for building a shared understanding of NFRs in CSE, we observe that CSE organizations are *proactively* building a shared understanding. However, we observed that CSE organization are still reactively building a shared understanding in several instances. In addition, we discuss the impediments to a shared understanding of NFRs in CSE, highlighting the similarities and differences between the impediments described in our study and prior research.

Furthermore, we discuss the benefits of CSE practices for building a shared understanding of NFRs. One of these benefits is continuous verification that encourages developers to build a solid understanding of NFRs like TESTABILITY to ensure that new bugs are likely not introduced in software during continuous integration activities. However, we discuss the drawbacks of CSE practices to building a shared understanding of NFRs such as the de-prioritization of NFRs due to the need for frequent release of software. Additionally, we discuss how CSE organizations can mitigate the

effects of remote collaboration through practices that improve communication and knowledge sharing such as sharing development standards.

Regarding the practical implications of our study, we hypothesize that Alpha can improve the shared understanding of NFRs when they invest in educating the software team on the value of NFRs to the quality and success of software. In addition, CSE organizations like Alpha can benefit from emphasizing NFRs by continuously testing and verifying NFRs, leading to developers having more understanding and experience with NFRs. Furthermore, we acknowledge the limitations of our research as a single-case study; however, we believe that our research adds valuable empirical evidence to this research area and is a helpful starting point to explore further how CSE organizations can build a shared understanding of NFRs.

In future research, we seek to evaluate the effectiveness of proactive practices for building a shared understanding of NFRs in CSE. In addition, we hope to explore the direct impact of continuous activities on the shared understanding of NFRs, mainly how other continuous activities not explored in our study, such as continuous evolution, influences the shared understanding of NFRs. Furthermore, we hope to investigate how CSE organizations can build and maintain a shared understanding of NFRs that have not been explored in this study, such as reliability or scalability. Finally, we hope that our research motivates more researchers to study how CSE organizations can mitigate the impediments to building a shared understanding of NFRs, specifically practices that can encourage the prioritization of NFRs.

# Bibliography

[1] Steve Adolph, Wendy Hall, and Philippe Kruchten. Using grounded theory to study the experience of software development. *Empirical Software Engineering*, 16(4):487–513, 2011.

[2] Par J. Agerfalk, Brian Fitzgerald, Helena Holmstrom Olsson, Brian Lings, Bjorn Lundell, and Eoin Ó Conchúir. A framework for considering opportunities and threats in distributed software development. 2005.

[3] Khaldoun Aldiabat and Carol-Lynne Le Navenec. Clarification of the blurred boundaries between grounded theory and ethnography: Differences and similarities. *Turkish online journal of qualitative inquiry*, 2(3):1–13, 2011.

[4] David Ameller, Claudia Ayala, Jordi Cabot, and Xavier Franch. How do software architects consider non-functional requirements: An exploratory study. In *2012 20th IEEE international requirements engineering conference (RE)*, pages 41–50. IEEE, 2012.

[5] SAIBS Arachchi and Indika Perera. Continuous integration and continuous delivery pipeline automation for agile software project management. In *2018 Moratuwa Engineering Research Conference (MERCon)*, pages 156–161. IEEE, 2018.

[6] Paul Atkinson. *Ethnography: Principles in practice*. Routledge, 2007.

[7] ATLAS.ti: The Qualitative Data Analysis & Research Software. `https://atlasti.com/`.

[8] Aruna D. Balakrishnan, Susan R. Fussell, and Sara Kiesler. Do visualizations improve synchronous remote collaboration? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1227–1236, 2008.

[9] Marianne Bamkin, Sally Maynard, and Anne Goulding. Grounded theory and ethnography combined: A methodology to study children's interactions on children's mobile libraries. *Journal of Documentation*, 2016.

[10] Kent Beck. *Extreme programming explained: Embrace change*. Addison-Wesley Professional, 2000.

[11] Woubshet Behutiye, Pertti Karhapää, Lidia López, Xavier Burgués, Silverio Martínez-Fernández, Anna Maria Vollmer, Pilar Rodríguez, Xavier Franch, and Markku Oivo. Management of quality requirements in agile and rapid software development: A systematic mapping study. *Information and software technology*, 123:106225, 2020.

[12] Alexander Bick, Adam Blandin, and Karel Mertens. Work from home after the COVID-19 Outbreak, 2020.

[13] Eva Alice Christiane Bittner and Jan Marco 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*, pages 106–114. IEEE, 2013.

[14] Jan Bosch. Continuous software engineering: An introduction. In *Continuous software engineering*, pages 3–13. Springer, 2014.

[15] Philip Cash, Elies A. Dekoninck, and Saeema Ahmed-Kristensen. Supporting the development of shared understanding in distributed design teams. *Journal of engineering design*, 28(3):147–170, 2017.

[16] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On non-functional requirements in software engineering. In *Conceptual modeling: Foundations and applications*, pages 363–379. Springer, 2009.

[17] Kevin R. Clark and Beth L. Vealé. Strategies to enhance data collection and analysis in qualitative research. *Radiologic technology*, 89(5):482CT–485CT, 2018.

[18] Cloud storage for work and home - Google Drive. `https://www.google.com/intl/en_ca/drive/`.

[19] Coda — The doc that brings it all together. `https://coda.io/`.

[20] Sharolyn Converse, J.A. Cannon-Bowers, and E. Salas. Shared mental models in expert team decision making. *Individual and group decision making: Current issues*, 221:221–46, 1993.

[21] Cucumber: BDD Testing and Collaboration Tools for Teams. `https://cucumber.io/`.

[22] Peter Darch, Annamaria Carusi, and Marina Jirotka. Shared understanding of end-users' requirements in e-science projects. In *2009 5th IEEE International Conference on E-Science Workshops*, pages 125–128. IEEE, 2009.

[23] Datadog: Cloud Monitoring as a Service. `https://www.datadoghq.com/`.

[24] Andrea De Lucia and Abdallah Qusef. Requirements engineering in agile software development. *Journal of emerging technologies in web intelligence*, 2(3):212–220, 2010.

[25] Adam Debbiche, Mikael Dienér, and Richard Berntsson Svensson. Challenges when adopting continuous integration: A case study. In *International Conference on Product-Focused Software Process Improvement*, pages 17–32. Springer, 2014.

[26] Advait Deshpande, Helen Sharp, Leonor Barroca, and Peggy Gregory. Remote working and collaboration in agile teams. 2016.

[27] Siva Dorairaj, James Noble, and Petra Malik. Effective communication in distributed agile software development teams. In *International Conference on Agile Software Development*, pages 102–116. Springer, 2011.

[28] Kevin Dullemond, Ben Van Gameren, and Rini Van Solingen. Virtual open conversation spaces: Towards improved awareness in a GSE setting. In *2010 5th IEEE International Conference on Global Software Engineering*, pages 247–256. IEEE, 2010.

[29] Kevin Dullemond, Ben van Gameren, and Rini van Solingen. Collaboration spaces for virtual software teams. *IEEE Software*, 31(6):47–53, 2014.

[30] Jonas Eckhardt, Andreas Vogelsang, and Daniel Méndez Fernández. Are non-functional requirements really non-functional? An investigation of non-functional requirements in practice. In *Proceedings of the 38th International Conference on Software Engineering*, pages 832–842, 2016.

[31] Omar Elazhary, Colin Werner, Ze Shi Li, Derek Lowlind, Neil A. Ernst, and Margaret-Anne Storey. Uncovering the benefits and challenges of continuous integration practices. *IEEE Transactions on Software Engineering*, 2021.

[32] Neil A. Ernst and Gail C. Murphy. Case studies in just-in-time requirements analysis. In *2012 Second IEEE International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 25–32, September 2012. ISSN: 2329-6356.

[33] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, 2017.

[34] Martin Fowler, Jim Highsmith, et al. The agile manifesto. *Software development*, 9(8):28–35, 2001.

[35] Gather - A better way to meet online. `https://www.gather.town/`.

[36] Cristina B. Gibson and Susan G. Cohen. *Virtual teams that work: Creating conditions for virtual team effectiveness*. John Wiley & Sons, 2003.

[37] GitHub: Where the world builds software. `https://github.com/`.

[38] Barney G. Glaser and Anselm L. Strauss. *The discovery of grounded theory: Strategies for qualitative research*. Routledge, 2017.

[39] Martin Glinz. On non-functional requirements. In *15th IEEE international requirements engineering conference (RE 2007)*, pages 21–26. IEEE, 2007.

[40] Martin Glinz and Samuel A. Fricker. On shared understanding in software engineering: An essay. *Computer Science-Research and Development*, 30(3):363–376, 2015.

[41] Gabriela Goldschmidt. To see eye to eye: The role of visual representations in building shared mental models in design teams. *CoDesign*, 3(1):43–50, 2007.

[42] Brett Heasman and Alex Gillespie. Neurodivergent intersubjectivity: Distinctive features of how autistic people create shared understanding. *Autism*, 23(4):910–921, 2019.

[43] Kaisa Henttonen and Kirsimarja Blomqvist. Managing distance in a global virtual team: the evolution of trust through technology-mediated relational communication. *Strategic Change*, 14(2):107–119, 2005.

[44] Pamela J. Hinds and Suzanne P. Weisband. Knowledge sharing and shared understanding in virtual teams. *Virtual teams that work: Creating conditions for virtual team effectiveness*, pages 21–36, 2003.

[45] Rashina Hoda. Socio-technical grounded theory for software engineering. *IEEE Transactions on Software Engineering*, 2021.

[46] Rashina Hoda, James Noble, and Stuart Marshall. Using grounded theory to study the human aspects of software engineering. In *Human Aspects of Software Engineering*, pages 1–2. 2010.

[47] Rashina Hoda, James Noble, and Stuart Marshall. Grounded theory for geeks. In *Proceedings of the 18th conference on pattern languages of programs*, pages 1–17, 2011.

[48] Yvonne Hsieh. Culture and shared understanding in distributed requirements engineering. In *2006 IEEE International Conference on Global Software Engineering (ICGSE'06)*, pages 101–108. IEEE, 2006.

[49] Jez Humble. Continuous delivery sounds great, but will it work here? *Communications of the ACM*, 61(4):34–39, 2018.

[50] Jan Ole Johanssen, Anja Kleebaum, Bernd Bruegge, and Barbara Paech. How do practitioners capture and utilize user feedback during continuous software engineering? In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 153–164. IEEE, 2019.

[51] Jan Ole Johanssen, Anja Kleebaum, Barbara Paech, and Bernd Bruegge. Practitioners' eye on continuous software engineering: An interview study. In *Proceedings of the 2018 International Conference on Software and System Process*, pages 41–50, 2018.

[52] Shawn Jordan and Robin Adams. Perceptions of success in virtual cross-disciplinary design teams in large multinational corporations. *CoDesign*, 12(3):185–203, 2016.

[53] Nahoko Kameo. A culture of uncertainty: Interaction and organizational memory in software engineering teams under a productivity scheme. *Organization studies*, 38(6):733–752, 2017.

[54] Michelle E. Kiger and Lara Varpio. Thematic analysis of qualitative data: AMEE Guide No. 131. *Medical teacher*, 42(8):846–854, 2020.

[55] Maaike Kleinsmann and Rianne Valkenburg. Barriers and enablers for creating shared understanding in co-design projects. *Design studies*, 29(4):369–386, 2008.

[56] Jonas Kniel and Alice Comi. Riding the same wavelength: Designers' perceptions of shared understanding in remote teams. *SAGE Open*, 11(3):21582440211040129, 2021.

[57] Robert E. Kraut, Robert S. Fish, Robert W. Root, and Barbara L. Chalfonte. Informal communication in organizations: Form, function, and technology. In *Human reactions to technology: Claremont symposium on applied social psychology*, pages 145–199, 1990.

[58] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[59] Rakesh Kumar Lenka, Srikant Kumar, and Sunakshi Mamgain. Behavior driven development: Tools and challenges. In *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pages 1032–1037. IEEE, 2018.

[60] Ze Shi Li, Colin Werner, Neil A. Ernst, and Daniela Damian. Towards privacy compliance: A design science study in a small organization. *Information and Software Technology*, 146:106868, 2022.

[61] Tatiana Losev, Sarah Storteboom, Sheelagh Carpendale, and Søren Knudsen. Distributed synchronous visualization design: Challenges and strategies. In *2020 IEEE Workshop on Evaluation and Beyond-Methodological Approaches to Visualization (BELIV)*, pages 1–10. IEEE, 2020.

[62] Dewi Mairiza, Didar Zowghi, and Vincenzo Gervasi. Conflict characterization and analysis of non functional requirements: An experimental approach. In *2013 IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT)*, pages 83–91. IEEE, 2013.

[63] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. Managing conflicts among non-functional requirements. In *Australian Workshop on Requirements Engineering*. University of Technology, Sydney, 2009.

[64] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. An investigation into the notion of non-functional requirements. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 311–317, 2010.

[65] Abderrahman Matoussi and Régine Laleau. A survey of non-functional requirements in software development process. LACL, 2008.

[66] Courtney Miller, Paige Rodeghero, Margaret-Anne Storey, Denae Ford, and Thomas Zimmermann. "How Was Your Weekend?" Software Development Teams Working From Home During COVID-19. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 624–636. IEEE, 2021.

[67] John Mylopoulos, Lawrence Chung, and Brian Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on software engineering*, 18(6):483–497, 1992.

[68] Tahereh Fathi Najafi, Robab Latifnejad Roudsari, Hossein Ebrahimipour, and Narjes Bahri. Observation in Grounded Theory and Ethnography: What are the Differences? *Iranian Red Crescent Medical Journal*, 18(11), 2016.

[69] Hidenori Nakai, Naohiko Tsuda, Kiyoshi Honda, Hironori Washizaki, and Yoshiaki Fukazawa. A SQuaRE-based software quality evaluation framework and its case study. In *2016 IEEE Region 10 Conference (TENCON)*, pages 3704–3707. IEEE, 2016.

[70] Yotam Ophir, Dror Walter, and Eleanor R. Marchant. A collaborative way of knowing: Bridging computational communication research and grounded theory ethnography. *Journal of Communication*, 70(3):447–472, 2020.

[71] Process Street — Checklist, Workflow and SOP Software. `https://www.process.st/`.

[72] Sadhana Puntambekar. Analyzing collaborative interactions: Divergence, shared understanding and construction of knowledge. *Computers & education*, 47(3):332–351, 2006.

[73] Margaret R Roller and Paul J. Lavrakas. *Applied qualitative research design: A total quality framework approach*. Guilford Publications, 2015.

[74] Ronnie E.S. Santos, Cleyton V.C. Magalhães, and Fabio Q.B. Da Silva. Member checking in software engineering research: Lessons learned from an industrial case study. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 187–192. IEEE, 2017.

[75] Ravi Sen and Sharad Borle. Estimating the contextual risk of data breach: An empirical approach. *Journal of Management Information Systems*, 32(2):314–341, 2015.

[76] Helen Sharp, Yvonne Dittrich, and Cleidson RB De Souza. The role of ethnographic studies in empirical software engineering. *IEEE Transactions on Software Engineering*, 42(8):786–804, 2016.

[77] Helen Sharp, Mark Woodman, and Hugh Robinson. Using ethnography and discourse analysis to study software engineering practices. In *Twenty-second International conference on software engineering*. Citeseer, 2000.

[78] N.C. Shrikanth, William Nichols, Fahmid Morshed Fahid, and Tim Menzies. Assessing practitioner beliefs about software engineering. *Empirical Software Engineering*, 26(4):1–32, 2021.

[79] Slack is where the future works. `https://slack.com/`.

[80] Darja Smite, Nils Brede Moe, Georgiana Levinta, and Marcin Floryan. Spotify guilds: how to succeed with knowledge sharing in large-scale agile organizations. *IEEE Software*, 36(2):51–57, 2019.

[81] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. Grounded theory in software engineering research: a critical review and guidelines. In *Proceedings of the 38th International conference on software engineering*, pages 120–131, 2016.

[82] Evenynke Terpstra, Maya Daneva, and Chong Wang. Agile practitioners' understanding of security requirements: Insights from a grounded theory analysis. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 439–442. IEEE, 2017.

[83] The Personal Information Protection and Electronic Documents Act (PIPEDA). `https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/the-personal-information-protection-and-electronic-documents-act-pipeda/`.

[84] Peter Törlind and Andreas Larsson. Supporting Informal Communication in Distributed Engineering Design Teams. In *International CIRP Design Seminar: 16/05/2002-18/05/2002*, 2002.

[85] Saeed Ullah, Muzaffar Iqbal, and Aamir Mehmood Khan. A survey on issues in non-functional requirements elicitation. In *International Conference on Computer Networks and Information Technology*, pages 333–340. IEEE, 2011.

[86] Colin Werner. Towards A Theory of Shared Understanding of Non-Functional Requirements in Continuous Software Engineering. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 498–503. IEEE, 2021.

[87] Colin Werner, Ze Shi Li, Neil A. Ernst, and Daniela 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)*, pages 90–101. IEEE, 2020.

[88] Colin Werner, Ze Shi Li, Derek Lowlind, Omar Elazhary, Neil A. Ernst, and Daniela Damian. Continuously managing NFRs: Opportunities and challenges in practice. *IEEE Transactions on Software Engineering*, 2021.

[89] Liang Yu, Emil Alégroth, Panagiota Chatzipetrou, and Tony Gorschek. Utilising CI environment for efficient and effective testing of NFRs. *Information and Software Technology*, 117:106199, 2020.

# Appendix A

# Semi-structured Interview Recruitment Documents

## A.1 Email to Organization

Dear CONTACT_NAME,

Thank you for taking an interest in my research study with the SEGAL research group entitled 'A Case Study of the Shared Understanding of software requirements in a midsize agile software organization' at the University of Victoria. As a graduate student, I am required to conduct research as part of the requirements for a degree in the Masters in Computer Science. This research is being funded by the Natural Sciences and Engineering Research Council of Canada (NSERC). It is being conducted under the supervision of PRINCIPAL INVESTIGATOR 1 and PRINCIPAL INVESTIGATOR 2. You may contact my supervisors at EMAIL_ADDRESS 1 and EMAIL_ADDRESS 2.

As previously discussed, the purpose of this research study is to explore the shared understanding of software requirements in a midsize agile software organization. The objective is to provide an exposition into the individual perspective of the shared/mutual understanding of software requirements within a collaborative software engineering team and evaluate how specific organization practices may influence this perspective. Your organization could benefit from this study by understanding why a shared understanding of software requirements is important and how it affects the software team's role and the delivery of software products within your organization.

Sequel to our previous meetings, we were hoping you could introduce our research

study to the software engineering team within your organization, informing them of our ethical considerations described in this email. As a fulfilment of the ethics requirements set by the Human Research Ethics Board at the University of Victoria, the participation of your employees in our research study must be voluntary and an option to not participate in this study can not result in any consequences to the respective candidate. Your employees who choose to participate in this study would be required to give their consent, and they can withdraw their consent to participate in this study at any time without consequence, and we would delete any data collected permanently.

If your employees consent to voluntarily participate in this study, their participation will include participating in formal/informal interviews, surveys and allowing a non-participative observation of informal/formal meetings within the software engineering team. The interviews are aimed to explore participant perspectives on shared understanding, how they maintain shared understanding when working collaboratively on a project, and how your organization's practices have helped them reach shared understanding when working collaboratively on a project. The interviews will take place virtually and data would be recorded collected through Zoom. The non-participative observations involve the researcher observing how the participants collaborate with the team when working on projects, how they document and resolve project issues, and how they reach a mutual understanding of work to be done. Notes would be taken during the non-participative observations excluding personal or confidential information that should not be shared outside of the meeting or organization.

In terms of protecting your anonymity, pseudonyms would replace all identifiable information in the data collected. Data collected would not be shared with anyone outside of the research team, and result findings would not include any identifiable information of your employees who choose to participate in this study. Additionally, there are no limits to the anonymity of the participants, including during the non-participative observations. During observations, we would exclude any identifiable information of the participants from the notes taken or replace them with pseudonyms. The researchers would not inform your organization of the employees that choose to participate in this study, and your organization would not have access to any data recorded that includes any identifiable information.

I have included copies of the invitation letter and consent forms we intend to use in the recruitment process for this study. Please reach out if you have any questions.

## A.2 Invitation to Participate

Hi CONTACT_NAME,

You are invited to participate in a study entitled 'A Case Study of the Shared Understanding of software requirements in a midsize agile software organization' that is being conducted by the SEGAL research group, University of Victoria.

As a graduate student, I am required to conduct research as part of the requirements for a degree in the Masters in Computer Science. It is being conducted under the supervision of PRINCIPAL INVESTIGATOR 1 and PRINCIPAL INVESTIGATOR 2. You may contact my supervisors at EMAIL_ADDRESS 1 and EMAIL_ADDRESS 2. This research is being funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

The purpose of this research project is to explore the shared understanding of software requirements in a midsize agile software organization. The objective is to provide an exposition into the individual perspective of the shared/mutual understanding of software requirements within a collaborative software engineering team and evaluate how this perspective may be influenced by specific organization practices.

You are being asked to participate in this study because you actively work collaboratively in teams to deliver software products for your organization (COMPANY_NAME). You could benefit from this study by gaining a deeper understanding of why a shared understanding of software requirements is important and how it affects your role and the delivery of software products within your organization.Your participation in this study is voluntary and choosing not to participate or withdraw from the study would create no consequences for you.

In terms of protecting your anonymity, pseudonyms would replace all identifiable information in the data collected. Data collected would not be shared with anyone outside of the research team and result findings would not include any identifiable information of the participants. There are no limits to the anonymity of the participants including during the non-participative observations. During observations, identifiable information of the participants would be excluded from the notes taken or replaced by pseudonyms. Additionally, this means that your organization would not be informed of your participation in this study and would not have access to any data recorded that includes any identifiable information.

Please reply to this invitation if you are interested in participating in this study, thereafter, a consent form would be shared with you to begin the study process.

## A.3 Signed Consent

University of Victoria

# *Participant Consent Form*

**A Case Study of the Shared Understanding of software requirements in a midsize agile software organization**

You are invited to participate in a study entitled 'A Case Study of the Shared Understanding of software requirements in a midsize agile software organization' that is being conducted by the SEGAL research group, University of Victoria.

Daniela Damian is a professor and faculty member in the department of Computer Science at the University of Victoria, and you may contact her if you have further questions by {EMAIL_ADDRESS}.

I am Laura Okpara, a graduate student in the department of Computer Science at the University of Victoria. As a graduate student, I am required to conduct research as part of the requirements for a degree in the Master's in Computer Science. It is being conducted under the supervision of Daniela Damian and Adam Murray. You may contact my supervisors at {EMAIL_ADDRESS} and {EMAIL_ADDRESS}.

This research is being funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

**Purpose and Objectives**
The purpose of this research project is to explore the shared understanding of software requirements in a midsize agile software organization. The objective is to provide an exposition into the individual perspective of the shared/mutual understanding of software requirements within a collaborative software engineering team and evaluate how this perspective may be influenced by specific organization practices.

**Importance of this Research**
Research of this type is important because shared understanding is crucial to the success of software organizations, and it is not well understood or prioritized by software engineering teams. A significant gap in shared understanding of software requirements can lead to an avoidable loss of time, money or resources. Research on shared understanding is fast-growing and the benefits can be realized by practitioners in agile software organizations. Additionally, this research aims to uncover any factors that may be hindering or limiting the shared understanding of software requirements within collaborative software engineering teams. We also aim to recommend to practitioners, including the participants of this study, practical strategies tailored to aid practitioners to build, manage and maintain a shared understanding of software requirements within collaborative teams.

**Participants Selection**
You are being asked to participate in this study because you actively work collaboratively in teams to deliver software products for your organization ({COMPANY_NAME}). Your participation in this study is voluntary and choosing not to participate or withdraw from the study would create no consequences for you.

**What is involved**
If you consent to voluntarily participate in this research, your participation will include participating in formal/informal interviews, surveys and allowing a non-participative observation of your informal/formal meetings with other members of your team. The interview is aimed at exploring your perspective on shared understanding, how you maintain shared understanding when working collaboratively on a project, and how

your organization practices has helped you reach shared understanding when working collaboratively on a project. The interviews would take place virtually. Data would be collected through note taking and through recording via a video conferencing technology, i.e., Zoom. The interviews would be expected to take a minimum of 30 minutes of your time and where there is a need for a follow-up interview or survey, this consent would be used unless it is withdrawn by the participant. The non-participative observations involve the researcher observing how you collaborate with your team when working on projects, how you document and resolve project issues, and how you reach a mutual understanding of work to be done. Notes would be taken during the non-participative observations excluding personal or confidential information that should not be shared outside of the meeting or organization.

*Please be advised that information about you that is gathered for this research study which does not include any identifiable information uses an online program (Microsoft forms) located in the U.S.A. As such, there is a possibility that information about you may be accessed without your knowledge or consent by the U.S. government, in compliance with the U.S. Freedom Act.*

**Inconvenience**
Participation in this study may cause some inconvenience to you, including the time you dedicate to participate in the interviews and surveys. The identification of participants would be anonymized and the responses from the interviews, surveys and notes from the observations would be kept confidential. Identification of participants would not be shared with anyone outside of the research team; however, results from the findings may be shared as a report or recommendation to the participants and the software organization, {COMPANY_NAME}. Reports or recommendations shared with the organization would not contain any identifiable information of the participants and as such, would not create a risk for the participants of this study.

**Risks**
There are no known or anticipated risks to you by participating in this research.

**Benefits**
The potential benefits of your participation in this research include:

1. This research contributes to the body of knowledge of shared understanding in agile software organizations, particularly highlighting factors that hinder shared understanding and practices that aid in building shared understanding.
2. The participants would benefit from this study by gaining a deeper understanding of why a shared understanding of software requirements is important and how it affects their roles and the delivery of software products.
3. Practitioners from midsize agile software organizations, including {COMPANY_NAME}, can benefit from the recommended practical strategies from our results to improve shared understanding of software requirements within their teams and improve the software development process.

**Voluntary Participation**
Your participation in this research must be completely voluntary. If you do decide to participate, you may withdraw at any time without any consequences or any explanation by simply informing the researcher of your request to withdraw. If you do withdraw from the study, you will be asked if you would like us to use any of your data collected. If you do not reply to this enquiry or your response is negative, your data would be deleted and overwritten from the researcher's computer.

**On-going Consent**
To make sure that you continue to consent to participate in this research, I may request further interviews, surveys, or observation. I would remind you, the participant that your participation in this research study is voluntary and can be withdrawn at any time without any consequence and ask if you are willing to continue to

consent. If you, the participant agrees to it, the interviews/surveys would be conducted without the need to sign another consent form, unless where the consent is withdrawn.

**Anonymity**
In terms of protecting your anonymity, pseudonyms would replace all identifiable information in the data collected. Data collected would not be shared with anyone outside of the research team and result findings would not include any identifiable information of the participants. There are no limits to the anonymity of the participants including during the non-participative observations. During observations, identifiable information of the participants would be excluded from the notes taken or replaced by pseudonyms. Additionally, this means that your organization would not be informed of your participation in this study and would not have access to any data recorded that includes any identifiable information.

**Confidentiality**
Your confidentiality and the confidentiality of the data will be protected by storing and encrypting the data collected on password-protected computers in the researcher's lab electronically. Identifiable information from the data collected would not be shared with anyone outside of the research team at any time.

**Dissemination of Results**
It is anticipated that the results of this study will be shared with others in the following ways:
Anonymized excepts from the interviews may be used in the Thesis/Dissertation.
The anonymized data collected, coded or analyzed may be used in the Thesis/Dissertation.
The results from this study may be used in a publication for a conference.
The results from this study would be used in the Thesis/Dissertation.
The results from this study may be shared with your organization, {COMPANY_NAME}, in the form of recommendations or reports.
If the results from this study is used in a publication, this would be available to the public, including your organization, {COMPANY_NAME}.

**Disposal of Data**
Data from this study will be stored for as long as we need it for this study but no longer than 3 years. Data from this study will be disposed by deleting and overwriting the data from the researcher's computer.

**Contacts**
Individuals that may be contacted regarding this study include the researcher, Laura Okpara, {EMAIL_ADDRESS}, the co-investigators/supervisors, Daniela Damian, {EMAIL_ADDRESS} and Adam Murray, {EMAIL_ADDRESS} .

In addition, you may verify the ethical approval of this study, or raise any concerns you might have, by contacting the Human Research Ethics Office at the University of Victoria {EMAIL_ADDRESS}.

Your signature below indicates that you understand the above conditions of participation in this study, that you have had the opportunity to have your questions answered by the researchers, and that you consent to participate in this research project.

| _____ | _____ | _____ |
|---|---|---|
| *Name of Participant* | *Signature* | *Date* |

***A copy of this consent will be left with you, and a copy will be taken by the researcher.***

# Appendix B

# Semi-Structured Interviews - Phase One

## B.1 Interview Questions

| Q# | Question |
|---|---|
| Q1 | How would you describe your role and the work you do at Alpha? |
| Q2 | How much experience do you have in your role in general and how much experience do you have at Alpha? |
| Q3 | What are the challenges you've experienced working collaboratively with other people on projects at Alpha? |
| Q4 | Do you recognize the difference between functional and non-functional requirements in software projects? |
| Q5 | Based on your role and experience, which non-functional requirements are vital to the success of your projects at Alpha? |
| Q6 | Have you heard of the term shared understanding? what does a mutual or shared understanding mean to you? |
| Q7 | How would you describe the process of reaching a shared/mutual understanding of non-functional requirements for any project? |
| Q8 | When can you say that everyone on your team understands the non-functional requirements sufficiently to produce substantial work? |
| Q9 | Do you recall when you or someone on your team/group did not fully understand the non-functional requirements of a project? What was the outcome of that experience? |
| Q10 | How can you tell that you have a comprehensive and accurate understanding of the non-functional requirements for a project? How can you tell that you don't? |
| Q11 | When working collaboratively on a project, how can you tell when someone on your team/group does not fully understand the non-functional requirements? |
| Q12 | When working collaboratively on a project, how can you tell when everyone on your team has a sufficient understanding of the non-functional requirements? How would you resolve this lack of shared understanding? |
| Q13 | How would you describe the factors that hinder or limit your understanding of non-functional requirements when working with other people? |
| Q14 | How would you describe the practices within Alpha that have encouraged a better mutual understanding of non-functional requirements? |
| Q15 | Who is responsible for ensuring that everyone on the project has a comprehensive and accurate understanding of the non-functional requirements? How is that responsibility shared? |
| Q16 | What communication methods are most effective for reaching a mutual understanding of non-functional requirements? |
| Q17 | What tools or techniques within Alpha have enabled you to reach a clear understanding of non-functional requirements more easily? What tools do you think would be relevant but currently not employed within your organization? |

Table B.1: Interview questions used in the first phase of interviews

## B.2 Codebook

NFR - Non-functional requirements

| Code | Code name | Code Description | Example |
|---|---|---|---|
| Code1 | NFR: Performance | Participants describe performance as it relates to responsiveness and load time. Includes load time, responsiveness, response time, quick or fast response. | On the kickoff, they wanted to know exactly how fast our server was able to turn around responses when they made API calls. |
| Code2 | NFR: Maintainability | Participants describe maintainability as it relates to developers or other stakeholders being able to understand or maintain aspects of a product. Includes writing understandable code, includes processes that improve developer experience. | But you could definitely consider there to be non-functional requirements for our public API's. And the things that you want to consider in those cases are, does the API make sense? Is it familiar for other developers? Is it something that they could easily discover? If they were to look at the schema? |
| Code3 | NFR: Usability | Participants describe usability as it relates to the visual aspects of a product. Includes aesthetics, look and feel, the visual aspects of a product, user experience, arrangement of visual components. | When you're dealing with visual specs, the most important thing is like, is usually how it behaves in different environments. |
| Code4 | Validating/Capturing NFRs | Participants elicit/validate non-functional requirements through a variety of feedback cycles. During the feedback cycle, the development team can validate if an NFR is being met or elicit new NFRs that haven't been implemented/considered yet. Feedback cycles include code reviews, meetings, informal conversations geared at giving or receiving feedback and writing/reading specifications. | But it's that idea to me about like a lot of the non-functional requirements, they come out of either using the product or understanding the customers really nuanced needs. And so we might bring someone in to get an understanding of what the client is saying because people just understand it so much more when either they can see it themselves, or they can hear it right from the customer's mouth. |
| Code5 | Unspecified NFRs | Participants report that the non-functional requirements are mostly unwritten or unspecified. Includes keywords: undocumented, not documented, unspoken or unwritten, not organized, unclear or misunderstood, etc. | So like things like usability or like, like aesthetics or like look and feel of it, those aren't really documented or recorded or as, or as organized as the big picture things. |

| | | | |
|---|---|---|---|
| Code6 | Discussing problems and solutions | Participants resolve issues, misunderstandings or changes in the scope of projects through discussing problems and solutions. Includes different types of meetings, informal conversations about problems or solutions. (must be specific to NFRs) | Well, what I'll do is, I'll set a meeting with everybody involved in the projects. And if something changes, I'll make sure that everybody who is involved is together, and then we can actually hash it out and talk and then I'll usually take some kind of notes. |
| Code7 | Asking questions | Participants use question asking to reach a mutual understanding of a goal. Includes asking questions during formal and informal meetings, includes any other reference to asking questions to reach a mutual understanding. | And you have to ask questions and make sure that the person fully understands what the goal is. Sometimes things can be unclear, so long as you're constantly asking questions and the person is confident that they know what they're doing. |
| Code8 | Abstracting solutions through visual aids | Participants abstract solutions through visual aids to resolve issues and solve problems. Includes sketching or designing possible solutions using a variety of tools. | A lot of the times when we are kind of faced with a problem, we use figma jam as like a real time way to like, kind of hash IT solutions and see how stuff might look. So I think that's like kind of an important one too. |
| Code9 | Individual Differences | Participants report differences in individuals as a challenge for reaching mutual understanding. Includes differences in points of view, ideas, skills, experience, background and level of knowledge within the organization. | The first thing that comes to mind when working with other team members, especially in a leadership position, is understanding that not every team member has the same level of technical or work experience as any other team member. |
| Code10 | Learning NFRs through experience | Participants describe learning about non-functional requirements through experience within the organization. Includes limitations to knowledge due to lack of experience, learning how people value NFRs, learning about NFRs over time. | Well, definitely the non-functional requirements have been like I mentioned earlier, its sort of the way that you learn loads is by running into, like, work pretty much like you have to figure out how to do something. So you ask, and then you learn what the requirements are for these projects.<br><br>—----------<br>I think the biggest limitation has just been like lack of experience, in my experience, so not knowing what a good cucumber spec is, until I encountered one. And then, you know, learning how to actually implement them. |

| | | | |
|---|---|---|---|
| Code11 | Customer context | Participants describe limitations from clients/customers as one of the reasons for unclear or undocumented NFRs. Includes unclear requirements from customers, insufficient elicitation of NFRs from customers, understanding the customers, understanding customer's need, understanding the origin of customer projects, learning about customers. | I'd say, like, a challenge can be extrapolating that like, small set of requirements to what the larger set of requirements should be, if that makes sense, right? A customer might come and say I, you know, want this button to do a certain thing, right. But then there's like a lot of undocumented requirements that involve that button, or that piece of functionality. |
| Code12 | Gaps in communication | Participants describe gaps in communication between stakeholders that could lead to a change in the scope of projects or other problems. Includes when there is a lot of feedback during meetings, when people make assumptions about a product or process, when people make assumptions about a person's level of knowledge/experience, it could lead to misunderstanding within the team. | There have been times where I've had a conversation with, with Logan about what a product needs to do and how he has envisioned it. And I guess some things just maybe got lost in translation. And I end up working on something that is different, or works differently than what he had imagined. And vice versa. |
| Code13 | Common grasp of problems/solutions | Participants describe their perspective of 'shared understanding'. Includes agreement, being on the same page, understanding within the team and any other references to mutual agreement or understanding. | That would mean to me is like, it's it's like everyone being on the same page with process, what the project is like understanding and how we're going to get there as well. |
| Code14 | Limitations on team's abilities. | Participants describe limitations on the abilities of developers or other stakeholders. Includes limitations on skills, experience, workload, background. | Just because a lot of times I am the only designer. So it is a little bit difficult sometimes to get feedback from people on my designs. And everyone on the team is really busy, of course. So it is difficult to get a hold of them sometimes to get their thoughts on it really quickly and as often as I needed. |
| Code15 | Increased awareness | Participants describe increased awareness on shared understanding and NFRs. Includes better understanding reached as a result of the interview sessions, increased | And lately, especially, I think that we've been shifting more towards focusing on the non-functional aspects. And this like, interview and like the work that you've been, like, talking about and doing has kind of, like, put a name to that and like, it's kind of like |

| | | | |
|---|---|---|---|
| | | understanding about NFRs, thinking about shared understanding or NFRs. | brought it to light, like, Okay, we need to, like work on these things and work to figure out how to measure and like, quantify and like, deal with understanding what we need to do for the non-functional aspects too |

# Appendix C

# Semi-Structured Interviews - Member Checking

## C.1 Interview Questions

| Q# | Question |
|---|---|
| Q1 | How is Gather different from other media that offer text+video channels like Zoom or Slack? |
| Q2 | How is working remotely through Gather comparable to working in person or face-to-face, from your experience at Alpha? |
| Q3 | How has the design guild helped your understanding of user experience and learning how to meet that requirement in the work that you do? Can you provide examples for this? |
| Q4 | How effective is wireframing interface design (using visual aids) when discussing or working on the user experience requirements of a product? Can you give examples of this? |
| Q5 | How effective are code reviews and the deep dive meetings for validating the user experience requirements of your designs? Can you provide examples for this? |
| Q6 | When the scope of a project changes or the project gets redesigned, what conversations happen among the stakeholders involved afterwards? How often does this occur? |
| Q7 | How do you usually learn about the expectations or standards for the user experience requirements of your designs? or specification writing? Can you provide examples or scenarios? |
| Q8 | How do you manage configuration and deployment during release management? Can you provide examples for this? |
| Q9 | How has the writers guild helped you to effectively elicit and document software quality requirements from stakeholders/customers (for instance, during journey mapping/data mapping)? Can you provide examples for this? |
| Q10 | What is the process for helping developers learn about the expected software quality requirements for a product from the stakeholders f? Can you provide specific examples for this? |
| Q11 | How effective are deep dive meetings for validating that the work in progress meets the software quality standards for testing, security and maintainability? Can you provide examples? |
| Q12 | As a guild leader for the mid stack, back-end, infrastructure and mobile development guilds, how do you help developers learn about software quality requirements standards specifically for software testing, writing specifications, security and maintainability of code-bases? Can you provide examples for this? |

Table C.1: Interview questions used in the member checking validation process

## C.2 Codebook

| s/n | CodeName | Code description | Examples | frequency |
|---|---|---|---|---|
| 1 | Asking questions | Participants use question asking to reach a mutual understanding of a goal. Includes asking questions during formal and informal meetings including any other reference to asking questions to reach a mutual understanding. | And then just asking the people who, like pretty much the developers above me, like Johann Logan, Sam, and Scott are all very knowledgeable. And so asking them for guidance is another source of it. | **2** |
| 2 | Eliciting NFRs through mockups | Participants use mockups to elicit some NFRs from customers such as user experience requirements. Mockups include: diagrams, sketches, wireframes, MVPs, demos, etc. Excludes scenarios unrelated to NFRs. | So you know, they'll say, we usually give them the widget and or the hosted portal, in kind of a default state. And then they come back to us and they say, Oh, can we change this? Can we change that? Can we move this change to this color, and then we work with the devs, to get it done. So it's a very sort of ad hoc process at the moment. | **12** |
| 3 | Gaps in knowledge | Participants describe gaps in knowledge that may exist when they do not have enough background or context about a product or customers or their organization. Includes lack of background knowledge, lack of knowledge, knowledge gaps, etc. | I think because I don't think I had enough background knowledge on exactly what the requirements were like the use cases were for that tag or the packages. And because it was a small part of a very large project, there wasn't any sort of, like kickoff, or anything or any sort of like a meeting or to talk about that small aspect of it. | **4** |
| 4 | Documenting solutions with diagrams. | Participants use diagrams, sketches or wireframes to document solutions during formal or informal meetings through virtual drawing or design tools. | So we had the conversation, impromptu conversation really, that came from talking about going over Colton designs where we are talking about and we're like, hey, like this doesn't really work, we need to kind of figure out exactly how we want to do these forms. Then Logan came over and he pulled out like a virtual drawing board and just we like documented okay for in this case, in these cases, the forms will look like this. These are the standards that we are going to use, like going forward with forms. | **6** |
| 5 | Validating NFRs through feedback. | Participants discover, capture and sometimes validate NFRs through feedback mechanisms. Feedback include user testing, code | user testing, behavioral user testing, which is giving somebody a user experience and seeing if they can make it to the end if they can make it to the goal, which will capture whether or not things are in the right place. | **21** |

| | | | | |
|---|---|---|---|---|
| | | reviews, deep-dive meetings and other meetings that help the developers validate or capture NFRs. | | |
| 6 | Deepening the understanding of NFRs | Participants describe ways they deepen/improve their understanding of non-functional requirements/learning about how to handle NFRs such as through specialized training, wireframing, code reviews, guilds (learning from guild members, learning through guild meetings), learning from previous projects/specifications, learning about the organization, etc. | But honestly, it comes down a lot to peer review. And someone saying, Hey, I wrote this test suite. Can you think of any other cases or have I missed something? And oftentimes, you know, those kinds of test suites can take a lot of time to create. So you know, there's a lot of opportunity for talking to the person who's writing the test suite and seeing where they are and just reviewing what they've done and what they plan to do. And then, you know, providing some guidance along the way.<br>_____<br>But it is also something for the presenter, you know, forcing somebody to present helps them organize their own thoughts, and helps kind of improve their own understanding of it, by presenting it. | **9** |
| 7 | Defining development **standards** through guilds. | Participants define or set development standards through guilds. Development standards refer to coding, design, documentation, testing standards, etc. Excludes any development standards that are not related to NFRs. | for the hook Guild and the view guild, We have, I think, bi-weekly meetings, every two weeks, we have a meeting. And that's to go over emerging standards or emerging libraries that we want to adopt. And also Logan will sometimes organize an exercise, like 20 minutes, here's a task, we're all going to do it and then we're going to compare results and learn from there. | **9** |
| 8 | Easy conversations on Gather | Participants describe Gather as a medium for having easy, informal conversations with each other. Includes mentions of informal conversations, easy conversations, talking, etc through Gather. | I feel like, with Gather, it's a lot easier just to quickly talk to someone. Whereas Zoom meetings that's usually pre-planned and pre-scheduled. And there's a link that has to be sent over and whatnot. And like, it's like all the little things that kind of add up to making someone not really want to reach out and like schedule a call? | **10** |
| 9 | Connectedness through Gather | Participants describe Gather as a medium that creates some connectedness with the team. Includes mentions of connectedness, not feeling alone, etc through Gather. | So I feel like when you're in Gather because you can see an avatar of the person you're talking to, or the people that are even sometimes just working around you. I think for myself, this is a really nice personal kind of connectedness that it generates. | **6** |

| | | | | |
|---|---|---|---|---|
| 10 | Expanding conversations on Gather | Participants describe Gather as a way to easily expand conversations to include more people. Includes expanding conversations, jumping into conversations, etc. | So if somebody that you need to talk to is talking to somebody else that you need to talk to, great time to go and kind of jump into the conversation. But if somebody is in kind of a closed-door meeting, well, you kind of have a better sense that it is not the appropriate time. | |
| 11 | Change in scope due to rework of designs. | Participants describe situations where designs are re-iterated or changed during the development phases. Includes changes in view, changes in design, etc. | So there's been cases where in design, it will look good. But once we actually develop it in the view, and have it working on the webpage, and tried and tested and realise that it's actually not good. It's like, you like, you can see that it's like knots, not fluids, or, like not natural. So there has been cases where we would reiterate the design, and then change the view that like, if that helps. | **7** |
| 12 | Measuring emotion response as feedback | Participants describe a form of feedback measured as emotional responses. Participants may validate or evaluate NFRs such as user experience using this feedback. Includes visceral reactions, emotional responses, physical reactions, etc. | But you know, there's also this opportunity to go in and actually using a tool like full story or something that's doing like video recording, just kind of watch users using it. **And I think you can tell based upon like, how many rage quits you get, and how many like, you know, frantic moving around the screens, you can really get a sense of just how much they're, they're at peace with it.** | **5** |
| 13 | Cross-functional roles and communication through guilds | Participants describe guilds are enabling cross-functional roles and communication for decision making. Includes cross-functional teams, communication through guilds across several teams, etc. | So right now, there's, there's just, it's not a big enough team to really have defined guilds, I think it's good that we have that foundation. And as the team grows, I think it's going to be good for having that sort of, you know, delineation and defining the different roles. | **7** |
| 14 | Evaluating development standards through feedback | Participants describe scenarios that allow them evaluate some development standards through feedback from customers/ code reviews, or other feedback mechanisms. | But yeah, most of those are really internal. I don't know how much the customers are setting our standard. I mean, they're allowing us to evaluate our standard, in my opinion, like, is this up to the customer's expectation? And then if it's no that we, you know, we can go back and talk about that. | **10** |
| 15 | Learning release management (NFR) through specialized training | Participants describe special training sessions organized by the organization that focuses on release management (configuration, deployment, documentation, etc). These | And one of the really cool things I think, was the top of the with the release process right now is that there's these sessions, these training sessions that are given with, you know, these presentations where people can ask questions and learn. But they're recorded. | **6** |

| | | | | |
|---|---|---|---|---|
| | | trainings help developers and non-developers learn about several aspects of release management. | And so now we get to start to build a library of knowledge about each release. And so the next team member can go back and watch those and learn, like they were here when that release happened. | |
| 16 | Need to understand Customer context | Participants describe the importance of understanding customer's background, product, or ideas in order to elicit more requirements from them. | And I really need to get into the head of the actual user that's going to use it. And I think that's when you start to learn things like how do they describe it, how, what are the unspoken requirements that they didn't really identify originally? And I think once you've got that information, then I can really move forward, but I need to really understand the problem from the eyes of the customer. | **3** |
| 17 | Defining standards for eliciting requirements | Participants describe how the team is defining standard architectures that help with eliciting requirements. | I literally just had a conversation with Mike this morning about defining standard architectures. And that's going to be a big thing moving forward is helping, like Beck and I have a lot of knowledge, having built these programmes over the past, you know, three or four years, and that is beneficial as a one person team | **2** |
| 18 | Communicating development standards with client | Participant describe communicating development standards with clients as a way to elicit requirements. | we've got all of those different, internationally recognised standards for everything from data retention to security, that we can then use to communicate with our clients. And, you know, they know, off the bat, if we say we are GDPR client compliant, then they know that well, that's good for their requirements, or maybe they need a different one. | **1** |
| 19 | Setting programme configurations with/for clients. | Participants describe the deployment and configuration process with clients. | So the basic idea, the basic flow is that you build out and fully configure the programme on test. And, you know, hook it up to any sort of tools that you have in tests. So a lot of clients will have, let's say, a Salesforce sandbox, or they'll have, you know, segments sandbox or Zapier or something. So they could completely build the programme out using test data, and then fully test it to make sure that it does exactly what they want. The events are triggered, you know, everything happens. | **3** |

| | | | | |
|---|---|---|---|---|
| 20 | Insufficient elicitations of NFRs | Participants describe a scenario where requirements were loosely acquired and this led to a lot of UX/design rework and a change in scope. | And it was very loosely, like, the requirements were very loosely acquired by will, during the sales process, we started to build it out. And they just kept coming back with all of these little changes, oh, they wanted a landing page, or they wanted to change the graphics. So they wanted to kind of go above and beyond what we would consider to be, you know, normal things that can be changed. | **1** |
| 21 | Sharing contents easily through Gather | Participant describe how Gather makes it easy to share content with other team members. | One thing that makes communication easy in gather compared to real life is you can share your screen easily since you're in front of your desk and the other person's in front of your that in front of their desk. So it's easy to like share contents as otherwise you just like go to your supervisors office and tell them I have a problem | **2** |

# Appendix D

# Publications

# A Case Study of Building Shared Understanding of Non-Functional Requirements in a Remote Software Organization

Laura Okpara, Colin Werner, Adam Murray, Daniela Damian
*Department of Computer Science*
*University of Victoria, Victoria, Canada*
{*lauraokpara, colinwerner, adammurray, danielad*}*@uvic.ca*

***Abstract*—Building a shared understanding of non-functional requirements (NFRs) is a known but understudied challenge in requirements engineering, especially in organizations that adopt continuous software engineering (CSE) practices. During the peak of the COVID-19 pandemic, many CSE organizations complied with working remotely due to the imposed health restrictions; some continued to work remotely while implementing business processes to facilitate team communication and productivity. In remote CSE organizations, managing NFRs becomes more challenging due to the limitations to team communication. While previous research has identified the factors that lead to a lack of shared understanding of NFRs in CSE, we still have a significant gap in understanding how CSE organizations, particularly in remote work, build a shared understanding of NFRs. We conduct a three-month ethnography-informed case study of a remote CSE organization. Through thematic analysis of our qualitative data from interviews and observations, we identify a number of practices for building a shared understanding of NFRs, such as validating NFRs through feedback. The collaborative workspace the organization uses for remote interaction is Gather, which simulates physical workspaces, and which our findings suggest allows for informal communications instrumental for building shared understanding. In addition, we describe the limitations to building a shared understanding of NFRs in the organization, such as gaps in communication and the limited understanding of customer context. As actionable insights, we discuss our findings in light of proactive practices that represent opportunities for software organizations to invest in building a shared understanding of NFRs in their development.**

***Index Terms*—shared understanding, continuous software engineering, non-functional requirements, remote**

## I. Introduction

Many software organizations are adopting continuous software engineering (CSE) practices, such as continuous integration and delivery [32], to support the release of working software through automation and shorter cycle times between releases [19]. CSE has implications for requirements engineering practices as it combines activities from agile methodologies that emphasize individuals and interactions, working software, customer collaboration, and response to rapid changes [20].

However, previous research has identified the lack of shared understanding as a significant challenge in managing non-functional requirements (NFRs) in CSE [55]. Shared understanding of NFRs is essential due to the complex, cross-cutting nature of NFRs [2] and the importance of NFRs to the success of software projects. For example, consider NFRs such as *security* [52] [47]; e.g. if data encryption is poorly implemented, a data breach can expose user information. Werner et al. describe the factors that contribute to the lack of shared understanding of NFRs, such as the fast pace of change and lack of domain knowledge [54]. However, there is still a lack of empirical evidence regarding how a CSE organization builds and manages a shared understanding of NFRs.

Working remotely creates further challenges in managing NFRs, along with their understanding, due to geographic distance, differences in context, and heavy reliance on communication technologies [22]. There are barriers to building a shared understanding, as conveying information through non-verbal communication is difficult in remote environments [35]. During the COVID-19 pandemic, there was a global shift from in-person activities to working remotely [7]. Many organizations had to adapt processes and systems to support team productivity, through virtual communication tools, social engagements, and peer support [43].

With the significant shift of software organizations working remotely, the motivation for this study stems from the need to explore how remote CSE software organizations build a shared understanding of NFRs to potentially minimize rework [54] and other costs that may result from a lack of shared understanding of NFRs. The primary goal of this study is to investigate how remote CSE software organizations build and maintain a shared understanding of NFRs. The secondary goal of this study is to provide additional insight into the limitations or challenges to a shared understanding of NFRs in CSE. Two main research questions guided our study:

**RQ1:** How does a *remote* software organization that adopts CSE practices reach a shared understanding of NFRs?

**RQ2:** What are the limitations to the shared understanding of NFRs in a *remote* software organization that adopts CSE practices?

Our work brings two significant contributions to the field of requirements engineering:

1) We add to the growing knowledge of shared understanding of NFRs in requirements engineering by explicitly describing practices for building a shared understanding of NFRs in remote organizations that adopt CSE practices.

2) We discuss practical ways for remote CSE organizations to proactively build a shared understanding of NFRs.

## II. Background and Related Work

### A. Non-functional requirements

Previous empirical studies suggest that managing NFRs can be challenging to software organizations for varying reasons, such as sparse documentation of NFRs or when documentation may be imprecise [2], along with the lack of consensus on how to sufficiently document and communicate NFRs [24]. NFRs are important to the success of software projects; however, research suggests that there is still some conflict in how software organizations manage NFRs [40]. NFRs are often neglected in requirements engineering due to the difficulties in eliciting NFRs, as NFRs have cross-cutting concerns with FRs [52]. Research suggests that NFRs have a representation problem and when NFRs are not described or represented properly, they can be misinterpreted as FRs [24]. NFRs such as *security* and *privacy* are important to software projects, but often neglected [52] and may be specified too late resulting in system vulnerabilities [47]. The main challenge is to understand the complex nature of NFRs and to create practices to manage the shared understanding of NFRs [41]. This is the gap that we intend to fill in this study by exploring practices to build a shared understanding of NFRs.

### B. Shared Understanding and CSE

CSE increases the need for an effective flow between customer needs and the rapid delivery of a product or service [19] through the automated release of working software in short release cycles. While this concept is already established through the agile manifesto [20], CSE is a more holistic approach as it includes activities to extend these practices such as continuous verification, improvement, and innovation [19].

Previous research suggests that shared understanding among people exists in two forms: implicit and explicit [25]. There is implicit shared understanding when people have a mutual understanding of unspecified facts or assumptions; whereas there is explicit shared understanding when people have a mutual understanding of specifications such as documents. Shared understanding is beneficial to CSE projects as it can promote successful collaboration [12]; when shared understanding exists, a team member is able to predict other's behaviours, there is an increase in team motivation and less conflict or mistrust among the team. Continuous integration, as CSE emphasizes, provides quick feedback to support the frequent release of working software, ensuring that potentially failure-inducing problems are identified and resolved as quickly as possible [19]. We bring further evidence from this study toward how shared understanding is built in CSE.

Werner et al. studied shared understanding of NFRs in CSE within three organizations [54]. They traced the effects of the lack of shared understanding in the form of rework in software projects, and they found that managing NFRs in CSE comes at a cost, such as fast pace of change, inadequate communication, and a lack of domain knowledge. They recommend practices for building a shared understanding of NFRs: communication and shared standards. Similarly, previous research reports the difficulties with automating NFRs and the problem of prioritizing functional requirements (FR) over NFRs [55]. However, these studies do not address how a CSE organization builds a shared understanding of NFRs.

### C. Shared Understanding and Informal Communication

Shared understanding cuts across several fields, disciplines, and contexts, such as education [45], medical science [27], design science [9], or engineering [25]. Several definitions of shared understanding exist in literature, one study describes shared understanding as "the ability of multiple agents to coordinate their behaviours with respect to each other to support the realization of common goals or objectives" [8]. Hinds conceptualizes shared understanding as a collective way of organizing relevant knowledge and a means for team members to anticipate and predict the behaviors of their peers or group [29]. These definitions focus on people and how they communicate to reach a common goal or objective. Research evidence suggests that in software organizations, informal communication is essential for understanding and communicating about stakeholder values and needs [6]. Informal communication is interactive and includes unplanned interactions that occur in the midst of daily activities [51] [16]. Kraut et al. discuss informal communication as based on the degree of preplanning, therefore, informal communication can be scheduled, intended, opportunistic, or spontaneous [36].

Research evidence in requirements engineering suggests that inadequate communication contributes to a lack of shared understanding [54]. Furthermore, daily informal communication allows team members to develop working relationships and encourages a better flow of information [1]. Having a virtual shared space in remote software organizations is important for collaboration on software projects [4]. In this study, we bring empirical evidence on *how* a remote software organization manages communication of NFRs in the absence of traditional in-person informal communication.

## III. Methodology

Our case study used ethnography-informed methods [3] [48] to study how a remote software organization, Alpha (fictitious name to protect confidentiality), attempts shared understanding in their product development. The ethnography-informed methods include observing participants at Alpha during their everyday work life and collecting different types of data related to how they build shared understanding while keeping an open mind for new discoveries and possibilities. In an ethnographic study, the researcher usually gains more understanding of the subject area over time to be able to identify and focus on more interesting or relevant aspects of the study [48]; however, our study was limited to three months.

In this section, we first outline the study setting in terms of our observations of the organization studied, its processes and working environment. We then follow with a description of the data collection and analysis methods.

### A. Study Setting

We identified a remote organization, Alpha, that practices CSE. Alpha is a Canadian based software organization that develops loyalty, rewards, and referral programs for businesses, as their primary revenue source. Alpha has operated for 9 years with 29 global employees across at least four time zones. Alpha is primarily client-focused, with most of its products having heavy visual components, including third-party platforms and storing client information. Alpha prioritizes NFRs such as performance, usability, security, and maintainability to meet their stakeholders' needs. The software engineering team at Alpha includes project leads (manager or developer), solutions architects, front-end and back-end software developers, user experience (UI/UX) designers/developers, and software integration engineers.

During the COVID-19 pandemic, Alpha shifted to remote work to comply with the health restrictions imposed on businesses within the region and country and, having adapted its business operations and approach, stayed fully remote for over two years. Alpha used Gather, a video chat platform designed to make virtual interactions more human.

*Alpha's communication methods and tools.* Similar to many other organizations during the COVID-19 pandemic, Alpha had to pivot to work remotely. Alpha initially used video conferencing tools such as Zoom; however, it soon discovered that it did not create connectedness within the software team, e.g. *"we started out like that when the pandemic first hit, we all went remote. Initially, it was so disconnected. No one felt connected to anyone else. It was so difficult to engage people in random conversation"* (Manager). Alpha then switched to using Gather [21], a virtual interaction tool that creates a work environment similar to a physical workspace, including desk spaces, meeting rooms, and avatars that represent the team members. Gather is fully customizable and allows Alpha to interact virtually as much as they would do in a physical space: whenever an avatar (team member) "walk" near another team member's avatar, a video call would be initiated in an easy and ubiquitous way, to simulate informal communication available in physical co-location, e.g *"Gather has made it possible for us as a remote team to have informal conversations. You literally just walk your little character up to someone else's little character, and you're talking to them."* (Manager) During meetings, Alpha's employees "walk" their avatar to the virtual meeting room in Gather and can choose to sit at the table. A group video call on Gather would begin as soon as they are near each other. Alpha's employees can also create private spaces (bubbles) with each other while in group meetings for one-on-one conversations when necessary, creating the feeling of being in the same space with each other. Alpha uses Gather to host social events, such as Christmas parties, encouraging team members to interact with each other beyond their daily work.

Alpha also uses Slack [49] to provide and receive frequent updates relevant to daily activities, and web documentation tools to communicate and document requirements such as Coda [11] to review backlogs, view the status of previous or ongoing tasks, and update tasks seamlessly throughout the life cycle of projects.

*Alpha's software and requirements engineering processes.* Alpha adopted CSE practices such as frequent feedback loops, continuous verification, and continuous testing based on stakeholder needs. Empirical evidence suggests an organization that adopts CSE may focus on certain CSE practices such as continuous integration for several reasons to suit their business and stakeholder needs [17]. The software engineering team at Alpha uses continuous delivery to frequently deliver working software in short cycles, welcoming changing requirements to meet the stakeholders' needs. In addition, Alpha implements a people-driven, autonomous approach incorporating tribes and guilds [50]. A tribe describes a group of people who work on similar feature areas of the software products; they coordinate and align themselves through the tribe leader, usually a senior member of the tribe. Tribe members have the necessary software engineering skills to come together and engineer working software features from "end-to-end." A guild describes a group of people with similar interests who have similar roles or perform similar tasks. Guild members come together to share knowledge, support each other and share processes to create a community that fosters learning and growth in specific areas.

Before starting a new project, a project kick-off meeting first occurs that includes stakeholders, such as success representatives, customers, or upper-level management. The meeting defines some of the high-level software requirements, including NFRs, and the project scope through user/data flows, product designs, and demos to understand the software product to be built. The project evolves with the delivery of working usable software in short cycles by continuously verifying project requirements with stakeholders throughout the software development process *"to ensure that the right thing has been built and it has been built right"* (Developer). The project lead initiates frequent feedback cycles such as QA, weekly or bi-weekly meetings with the stakeholders involved in the project; the feedback from these meetings can be change requests, new requirements or software bugs discovered. For instance, at Alpha, when application errors are not handled properly, it affects some important NFRs, such as the user experience, application reliability, and developers' ability to maintain the codebase easily (maintainability). The error handling issue is usually flagged as a bug and assigned to a developer working on the project. After the issue is resolved, it is reviewed and verified at a later time. The rapid feedback cycles continue and the project continues to evolve until all requirement requests have been implemented or all of the stakeholders are satisfied with the product built.

At Alpha, developers document software requirements as executable specifications using Cucumber [5], a test tool and framework that follows the behavior-driven design methodology (BDD) [38] to support the user and behavioural testing. At Alpha, developers have some freedom and flexibility to implement the defined and documented software requirements,

especially the NFRs. However, the software developers must collaborate with the project lead to remove ambiguity and create an architecture allowing all components to work together. The software engineering team at Alpha validates important NFRs such as performance, usability and maintainability. For performance requirements, Alpha monitors the performance of their APIs and applications through a third-party application performance monitoring tool, DataDog [13] to spool and visualize performance metrics such as error rates, count of application instances and average response times for API calls. Alpha validates usability through requesting feedback during demos, usability testing or behavioural testing with storybooks/stencilbooks by manually verifying the ability to reach specific pre-defined goals. Alpha also measures an *"emotional response score"* during usability testing with stakeholders by gauging their reactions to the demo of the intended product. Alpha validates other NFRs such as maintainability, testability and configuration management during code reviews, deep-dive meetings or daily download meetings where senior developers can review the written code to identify issues. Alpha uses automated test coverage tools to validate that unit tests written for specific codebases cover all possible functions identified within the code.

### B. Data Collection

For three months, one of the authors was embedded at Alpha to learn about the products, processes, and business. Through many informal conversations with the software engineering team, an intention was to understand how the team shared understanding, knowledge and how they communicate and document requirements. Alpha onboarded the on-site researcher by providing a work email, providing a dedicated computer, and granting access (through invitation) to their collaboration tools and platforms such as Gather, Slack, Coda, Google Drive [10] and GitHub [23]. Our data collection techniques were through observations and interviews.

*Participant Observation:* The on-site researcher observed the participants and their interactions during regular work activities and meetings with the software team and stakeholders, including customers and upper-level management. Alpha granted the researcher access to project documentation and repositories, which informed some of the researcher's observation notes. The researcher stored a collection of notes made during the observations and included them as a part of the memos used in the thematic analysis process. The researcher dedicated an average of 30 hours each week within Alpha during the three months of data collection. By engaging in informal conversations with employees and reviewing project documentation, the researcher learned about the roles and experiences of the employees and developed a working relationship with the team to help design the interview questions, describe the study setting adequately, and inform the interpretation of interview responses.

*Interviews:* Eleven semi-structured interviews were conducted via Zoom during the third month of the study. Each interview time varied between 30 and 45 minutes and was fully transcribed. The interviewees had various roles with varying years of experience, from a couple of months to fifteen years of experience. To protect the anonymity of our participants, we label developer/designer roles and team leads as developers and other remaining management roles as managers, as seen in Table I. Following a grounded theory approach, we iteratively developed the interview questions based on emerging patterns that helped us to establish an area of focus or interest [30]. The interviews focused on highlighting how shared understanding of NFRs is established and what challenges may limit the shared understanding of NFRs. We used open-ended questions to allow participants to describe their experiences. We sought each participant's view on shared understanding and how Alpha may reach a shared understanding of NFRs. The interviews were audio-recorded with consent. A sample of the interview questions can be found in Table II.

TABLE I
INTERVIEWEES

| Role | Work Area | Exp. in Org. | Overall Exp. |
|---|---|---|---|
| Developer | Visual Dev | <1 yr | <1 yr |
| Developer | Visual Dev | <1 yr | <1 yr |
| Developer | Project Engr | <1 yr | <1 yr |
| Developer | Visual Dev | <1 yr | <1 yr |
| Developer | Infrastructure | <1 yr | <3 yr |
| Developer | Infrastructure | <3 yr | <3 yr |
| Developer | Visual Dev | <3 yr | <3 yr |
| Manager | Product | <5 yr | <5 yr |
| Manager | Project/Product | <5 yr | <10 yr |
| Manager | Infrastructure | <5 yr | <20 yr |
| Manager | Executive | <10 yr | <10 yr |

TABLE II
SAMPLE INTERVIEW QUESTIONS

| Q# | Question |
|---|---|
| Q1 | How would you describe your role and the work you do at Alpha? |
| Q2 | Do you recognize the difference between functional and non-functional requirements in software projects? |
| Q3 | Based on your role and experience, which non-functional requirements are vital to the success of your projects at Alpha? |
| Q4 | Have you heard of the term shared understanding? what does a mutual or shared understanding mean to you? |
| Q5 | How would you describe the process of reaching a shared/mutual understanding of non-functional requirements for any project? |
| Q6 | When can you say that everyone on your team understands the non-functional requirements sufficiently to produce substantial work? |
| Q7 | How would you describe the factors that hinder or limit your understanding of non-functional requirements when working with other people? |

### C. Data Analysis

Data analysis involved fully transcribing audio recordings from interview sessions and creating insights using the information obtained through observations. The observation data collected was analyzed by closely examining the data to identify patterns and data relevant to our study, such as observation notes related to how developers collaborate when implementing NFRs. We used this data to describe our study setting

and design the research questions for the interviews. We also used the data from our observations to interpret the interview findings by understanding and describing Alpha's context and discussing our study's implications. We used the open and axial coding methods from grounded theory [30] to identify themes [34] across several codes from the interviews, while creating memos to describe the relationships and patterns observed in the data and any additional questions that may be explored. We also identified the triggers that led Alpha to build a shared understanding of NFRs, as suggested by Werner et al. [54], further discussed in Section V. Werner describes *triggers* as events that encourage an organization to invest in building a shared understanding [53]. Two independent coders were involved in the thematic coding process [34]. Agreement sessions were used to consolidate coding strategies, and to establish the reliability of the codes. We calculated the inter-rater reliability agreement until we reached substantial inter-rater reliability $>0.60$, using the Cohen Kappa's coefficient for measuring observer agreement for categorical data [37].

## IV. Findings

### A. RQ1: How does a remote software organization that adopts CSE practices reach a shared understanding of NFRs?

First, to understand how the software team views NFRs, we asked participants to tell us which NFRs were vital to the success of the projects that they work on. In our data analysis, we recognized three major NFRs that were mentioned repeatedly: usability, performance and maintainability. We also recognized a pattern where the NFRs mentioned during the interview sessions differed depending on the role and level of experience of the participant. Front-end developers and designers mainly describe usability and performance when referring to NFRs e.g. *"Well, usually the biggest NFRs that we get are everything to do with how it looks. When you're dealing with visual specs, the most important thing is usually how it behaves in different environments"* (Developer). Similarly, engineering leads and managers made reference to performance and understandability e.g. *"But you could definitely consider there to be NFRs for our public APIs. And the things that you want to consider in those cases are, does the API make sense? Is it familiar for other developers? Is it something that they could easily discover?"* (Manager). In another instance, a manager mentions security as an important NFR for their clients. e.g. *"They want to know that the data that they are providing us ... is secure, it's deleted when they need it to be deleted, and that the transmission is secure as well, that it's encrypted, and that there is no way to intercept the data as they're sending it to us or retrieving it"* (Manager).

We then asked participants how they reach an understanding of NFRs for any project they have worked on. From our data analysis, we identified five practices (themes from our data) at Alpha, outlined in Table III, and described below. In addition to the codes from which we developed these themes, Table III also provides information as to whether our data indicates that the shared understanding was *implicit* or *explicit* [25], whether it was the result of *triggers*, as found in the work of Werner and colleagues [54] and correspondingly whether they were *proactive* rather than *reactive*, i.e. no associated trigger:

- Validating NFRs through feedback
- Deepening the understanding of NFRs through experience
- Wireframing interface designs
- Discussing problems and solutions
- Asking questions

*1) Validating NFRs through feedback:* At Alpha, feedback is provided and received during the software development process, as enabled by their use of standardized communication tools such as Slack and Gather. A manager describes a scenario where during a demo meeting, performance was captured as a valuable NFR. e.g. *"I think we've actually seen this quite recently, with the demo project that's been happening here, where the engineering team was tasked with actually implementing the program. And by simply implementing it, you run up against all these NFRs that should exist. Because now you're going like, hey, I went to use it, it was really hard, I went to use it, it's taking forever to load"* (Manager).

Alpha benefits from the quick feedback loops of CSE in the form of daily meetings, other scheduled meetings and informal conversations that happen during the software development process. These meetings mostly happen through Gather, the virtual office, e.g *"We Slack or meet in Gather with screen-share and also every day, we have a morning meeting where we check-in and show what we're working on. And there can be opportunities for feedback there as well"* (Developer). In another instance, a software developer describes the project kick-off meeting that helps with discovering NFRs like extensibility *"Because when the 'shaping' is happening, we're starting to think about those NFRs about how it might work for other use cases"* (Developer). Similarly, our participants express that there are processes that encourage the validation and capturing of NFRs such as code review and other feedback methods. e.g *"Whereas the NFRs happen more passively. They're things that are built into processes, for example, when you create a pull request, there's a template that you have to fill out when you do a product launch ... And then obviously, there's things like code review, review and feedback from the managers that sort of catches the NFRs"* (Developer). In addition, a developer describes the channels of feedback as a way to actively validate NFRs like testability. e.g *"... And so if things are getting missed, like, tests or specs, that's an opportunity to get feedback, but really, it's any channel of feedback. So that could be slack messages, code review or Just conversations throughout the day on gather"* (Developer).

*2) Deepening the understanding of NFRs through experience:* Four of our participants expressed to us that they deepen their understanding of NFRs over time through the projects that they work on within Alpha. These NFRs are sometimes not explicitly defined but incorporated into already existing processes and the developer learns how to implement them through those processes over a period of time. e.g *"There's just a process that isn't formally presented, it's just picked up over time through training. I think that's the majority of*

TABLE III
SUMMARY OF PRACTICES FOR BUILDING A SHARED UNDERSTANDING OF NFRS (RQ1)

| **Practices** | **Description / Key Phrases** | **Implicit / Explicit** [25] | **Triggers** [54] | **Proactive / Reactive** |
|---|---|---|---|---|
| **Validating NFRs through feedback** | Validating NFRs through code review<br>Eliciting NFRs during feedback<br>Capturing NFRs from demos | Implicit | Rework | Proactive and Reactive |
| **Deepening the understanding of NFRs through experience** | Learning how the organization values NFRs<br>Learning about NFRs over time<br>Limitations due to lack of experience | Implicit/Explicit | Change in scope<br>Rework | Reactive |
| **Wireframing interface designs** | Sketching solutions<br>Designing solutions using visual aids | Explicit | N/A | Proactive |
| **Discussing problems and solutions** | Meetings for implementing NFRs<br>Informal conversations about problems | Implicit | Change in Scope<br>New requirements | Proactive and Reactive |
| **Asking questions** | Asking questions about NFRs<br>Asking questions formally or informally | Implicit | Change in scope<br>New requirements | Proactive and Reactive |

*NFRs are, for me anyways have been picked up that way"* (Developer). Developers told us that a helpful way to deepen their understanding of NFRs is through learning how NFRs are handled in previous similar projects. e.g. *"I think the biggest tool that has helped me learn NFRs has really just been reviewing previous work, reviewing previous specs, reviewing previous integrations, things like that, because it's the best way to get a sense of the expected code quality and expected way of doing things"* (Developer).

A developer describes how Alpha keeps people connected even while working remotely and how that helps with deepening the understanding of NFRs *"I did mention earlier that Alpha is remote first but I do find that they do a good job of keeping people connected. And I think Gather town has really helped with that, it's just a place where you can feel like you're next to the people that you're working with. And they're, a few button clicks away from a call to talk about things and screenshare"* (Developer). However, we observed that some of our participants expressed some difficulties in understanding NFRs due to a lack of experience. e.g *"I still think I'm learning what the requirements are for spec writing, because it's just it's a nuanced aspect of the requirements .... So it's something that I'm still learning, but definitely have a better understanding of than when I first started"* (Developer). At Alpha, developers document software requirements as executable specifications using a tool, Cucumber [5] that supports Behaviour-Driven Development (BDD). Our participants describe specification writing as an important NFR because it influences the degree to which an application supports testing and how application tests are written or understood. Nevertheless, a software developer describes learning Alpha's values as an important factor to gaining a shared understanding of NFRs e.g. *"I guess the tricky thing is that a lot of getting shared understanding, I think is almost kind of unwritten, it's like, you have to get a sense for how your organization values those NFRs to be able to kind of work with others around you"* (Developer).

*3) Wireframing interface designs:* We observed that another way Alpha reaches a shared understanding of NFRs was through wireframing. Our participants express that they mostly used Figma, sketches or minimal designs to describe some of the NFRs during project meetings or informal conversations. e.g. *"And that usually involves using something like figma, or fig jam to quickly visualize what we're talking about. So really basic forms, like shapes, and just text, and blocks of lines that connect different screens"* (Developer). To mitigate the challenges that could occur when working with visual components remotely, developers describe the use of screen-share on Gather, as a way that makes it easier to reach a shared understanding of NFRs. e.g *"A kind of advantage of Gather is that you get to share your screen and [person] likes to write diagrams and stuff like that or just mock-up code on the screen, which definitely makes things easier to understand"* (Developer). A manager further describes the effectiveness of wireframing interface designs by stating *"I think that's where things like 'shaping', wire-frames, starting to try to think through that user experience really helps. Design drawings, I find are really helpful. Early mock-ups help people because I find that a lot of the team does well with actually experiencing something. They quickly start to see ... this does the thing, but it's totally unusable for other reasons"* (Manager). For NFRs such as usability, visualizing what needs to be built and receiving feedback on those designs from other developers and clients ensures that there is a shared understanding of those NFRs. e.g. *"If we take the NFRs that are typically involved in something like visual design, in terms of user experience, obviously we have our designer who will design something, but we do try to obtain as much feedback on that design as possible. If possible, we actually like to communicate that or publicize it with the client, if there is something that's client-specific"* (Manager).

*4) Discussing problems and solutions:* Developers express that they sometimes gain more understanding about NFRs through discussions during regularly scheduled meetings on Gather, such as *bi-weekly guild meetings*. e.g *" It's really just a time to talk about how we work on things, which is a great idea. And it's been helpful for talking about how we build things and non-functional requirements in specific areas of our jobs"* (Developer). In another instance, a developer describes

an experience of how he ensures that everyone who works on a project understands the NFRs when there is a change in scope. e.g *"I'll set a meeting with everybody involved in the project. And if something changes, I'll make sure that everybody who is involved is together, and then we can actually hash it out and talk and then I'll usually take some kind of notes"* (Developer).

In addition, participants also describe their communication through Gather, as being key to facilitating several feedback cycles and meetings where it feels like they are working face-to-face. e.g. *"I can think of one software that we use, which is Gather. And Gather is really obviously a video communication tool. I think it does foster the ability for individuals to kind of interact more. And by interacting more, and by kind of stopping by somebody's desk that you may not have talked to in a while, you can kind of ensure that communication of different departments is kind of happening, and it's happening more organically"* (Manager).

*5) Asking Questions:* Through several feedback methods, we observed participants use question asking to reach a shared understanding of NFRs. A manager describes asking questions to clients to elicit the NFRs that are important to them during the early stages of discussing the project. e.g. *"you want to ask a lot of questions, you want to poke a lot of holes and things and, really go to the very edge of, you know, when you're sketching it out. So yeah, I would say that initially, it's a lot of questions"* (Manager). In a separate instance, a developer uses question asking to clarify an NFR and ensure that all members of the team have a good understanding of what the goal is. e.g Concerning usability, *"And you have to ask questions and make sure that the person fully understands what the goal is. Sometimes things can be unclear, but so long as you're constantly asking questions, then the person is confident that they know what they're doing"* (Developer).

Furthermore, we asked participants about what methods of communication have been effective for reaching a shared understanding of NFRs. All of our participants mentioned Gather i.e, *"And then we also have a virtual office environment called Gather, which is what we mainly use. I think Gather is good because it's more multimedia because you can talk and have video and share screens"* (Developer).

Finally, we asked participants when they know team members, inclusive of themselves, have a common or shared understanding of the NFRs. Developers describe this as when there is minimal feedback or questions during project meetings or conversations. e.g. *"Like when we go into the 'deep-dive' we know when the non-functional requirements have been met, when we as a group meet, and there aren't gaps in our shared understanding. We come out of a review meeting and there's not a lot of feedback"* (Developer). A manager describes this as when the team members are able to discuss the problems and solutions with each other and describe the behaviour of the software product e.g. *"I'm kind of sufficiently satisfied with two conditions, what I think is a little bit more subjective, which is when the team members are starting to be able to explain the problem and understand the detailed nuance explain that problem to others. The other one is when you start to see like acceptance criteria that's really starting to look like one of our actual user acceptance tests. And by that point, you get a sense of if someone really understood the non-functional"* (Manager). However, our participants acknowledge that shared understanding of NFRs should continue to evolve and there may not be a way to guarantee that there is a complete shared understanding of NFRs within the team at one time. A manager expresses this by saying *"I don't actually think you can have [guarantee], honestly, until it's actually in a client's hands being used or users hands. Because, I mean, there's a chance we got the NFRs wrong, you know, every single person did"* (Manager). In a different instance, a manager states that *"I don't think it's possible to reach 100% shared understanding, there's always going to be areas where people are running on assumptions, or people might have different interpretations"* (Manager).

### *B. RQ2: What are the limitations to a shared understanding of NFRs in a remote CSE software organization?*

We asked participants if they experienced any challenges with understanding NFRs when working collaboratively. From our data analysis, we found five main themes that describe the limitations to a shared understanding of NFRs: gaps in communication, limited understanding of customer context, individual differences and unspecified NFRs. Additionally, we identified the main triggers across these limitations. The themes, triggers, and associated codes are in Table IV.

*1) Gaps in communication:* In several instances, our participants describe gaps in communication within the software team and gaps in communication between the software team and the stakeholders due to unclear expectations for projects or assumptions about projects, which may lead to some rework or a change in the scope of a project. A manager describes a scenario in which the software team made inaccurate assumptions about what the stakeholders expected for a project regarding usability, which led to discussing additional requirements that were not within the scope of the project, *"They had made the assumption that they would be able to go in and update these prints after launch. And we had made the interpretation that everything was just sort of hard-coded, write once and forget it. And so, in the end, it was something that we actually had to push back on the client and say that, making that something that's editable, would significantly increase the scope"* (Manager). Another member of the software team describes a situation where there was some unclear communication regarding the project expectations for usability requirements which resulted in the developer working with inaccurate specifications. *"There have been times where I've had a conversation with [person] about what a product needs to do and how he has envisioned it. And I guess some things just maybe got lost in translation. And I end up working on something that is different or works differently than what he had imagined. And vice versa"* (Developer).

*2) Limited understanding of customer context:* A major challenge with reaching a shared understanding of NFRs when working with external customers is the limited understanding

TABLE IV
LIMITATIONS TO BUILDING A SHARED UNDERSTANDING OF NFRS (RQ2)

| Limitations | Description / Key Phrases | Triggers |
|---|---|---|
| **Gaps in Communication** | Making assumptions about a project/process<br>Unclear communication of expectations | Redesign/Reshape of projects<br>Scope creep<br>Change in scope |
| **Limited understanding of customer context** | Learning about customers<br>Insufficient understanding of customer needs | Scope Creep |
| **Individual differences** | Differences in ideas and background<br>Differences in skills and knowledge<br>Differences in approach to work | Rework |
| **Unspecified NFRs** | Uncommunicated NFRs<br>Unwritten NFRs<br>Unclear NFRs | Difficulty implementing NFRs<br>New requirements |

of the customer's business, products or needs. The lack of contextual knowledge about the customers can sometimes lead to misunderstandings or assumptions concerning the NFRs that are important to the customers, especially when an organization needs sufficient information about their customers or their customer's products to deliver a product successfully, as observed *"if I'm unable to understand what the motivating use cases are if I'm unable to see how this project actually came to be, what the scope is even, I'm unable to go dive into what those NFRs are"* (Developer).

Similarly, a participant expressed having experienced difficulty when working on a project and not understanding extensibility as an NFR. This difficulty was mainly due to a limited understanding of the customer and their products. e.g. *" I think a lot of the confusion probably also came from it being my first couple projects here. And so I was still wrapping my head around what our customers actually are, how we interact with them, the full range of our products, and how they're planning on implementing their programs"* (Developer).

*3) Individual differences:* Differences in the background, skills, experience and ideas of team members can sometimes lead to misunderstandings of NFRs, especially when there is no common foundation of knowledge. Alpha recognizes that having a diverse software team means that team members would have a different level of knowledge, skills and experience. e.g *"the first thing that comes to mind when working with other team members, especially in a leadership position, is understanding that not every team member has the same level of technical or work experience as any other team member"* (Manager). However, sometimes team members still make assumptions about how other members of the team think and how much they know and this may lead to a disparity in ideas about the product and how to implement the NFRs of the product. e.g. *"But it just trickled down to things like the language we're using, just wasn't aligned even, people didn't even have the same ideas about what the functionality was. And that just like continued to trickle through the other non-functional requirements because it wasn't meeting some of the use cases anymore"* (Developer).

*4) Unspecified NFRs:* We observed that some of our participants experienced difficulties with understanding or implementing NFRs due to unclear or unspoken expectations around these NFRs. Concerning maintainability, how a code-base can be understood and built upon with well-written documentation, a software developer describes difficulties. e.g *"Definitely, for the specification writing for that project, it was pretty difficult for me, because I wasn't really sure what they should look like. And also there weren't great examples that I could base it off"* (Developer). When NFRs are not clearly defined, communicated or documented, developers are often left with their assumptions on what NFRs to include for a project and how to implement the NFRs. Our participants expressed that this could happen, however, due to the frequent feedback loops within the software team, there are several opportunities to communicate and re-elicit the NFRs relevant to the work being done. e.g *"The thing is non-functional requirements are not always communicated as much... trying to figure out what or not, understanding or communicating expectations around some of those NFRs from stakeholders. The good thing about developing at Alpha is that our loops are pretty quick, we have a lot of communication, so becomes quite clear when something isn't being met. And when that comes up, just work to try and elicit what those requirements are"* (Developer).

## V. DISCUSSION

CSE advocates for ongoing customer feedback and rapid iteration cycles [19] [25], which should result in a higher level of understanding of customer requirements. However, prior research indicates that CSE may pose a detrimental effect on the shared understanding of non-functional requirements [54]. In remote organizations, building a shared understanding becomes more challenging due to the additional recognized barriers with remote work, such as challenges with information sharing and team members having a reduced ability to build trustworthy relationships [35].

Our work explores the delicate balance of achieving a shared understanding of NFRs in a remote CSE organization (Alpha), and in this section, we discuss the findings of our empirical study. Through our research, we uncovered five practices from

Alpha to build a shared understanding of NFRs: validating NFRs through feedback, deepening the understanding of NFRs through experience, wireframing interface designs, discussing problems and solutions, and asking questions. We found some limitations within Alpha to building a shared understanding of NFRs, such as gaps in communication that may occur when team members make assumptions about each other's knowledge. We discuss how Alpha mitigates some challenges with building a shared understanding in a remote setting through multiple collaboration tools, particularly Gather.

We believe these findings have important research implications concerning the intersection of shared understanding of NFRs and CSE in remote collaboration. For instance, how an organization could use the practices observed at Alpha, that are proactive, rather than reactive, to build a shared understanding of NFRs.

### A. Shared Understanding of NFRs in CSE

Glinz describes implicit shared understanding as the mutual understanding of non-specified facts or assumptions [25]. Our study suggests that Alpha relies on *implicit* shared understanding of NFRs, partly due to CSE practices that encourage frequent feedback [19] through formal, informal or face-to-face communication[14] [16], as observed *"a lot of getting to shared understanding is almost kind of unwritten"* (Developer). Our findings corroborate and bring additional, empirical detail about the amount of implicit shared understanding, as four of the five practices used to develop a shared understanding of NFRs were in fact implicit, namely validating NFRs through feedback, deepening the understanding of NFRs through experience, discussing problems and solutions, and asking questions. Only a single practice, wireframing interface designs, was explicit in nature.

At Alpha, one of the practices for building a shared understanding of NFRs is by validating NFRs through feedback. Our results show that developers at Alpha validate NFRs mainly through tests and manual validation through code reviews, *deep-dives* or daily downloads. Concerning some NFRs, such as testability, Alpha uses automated test coverage tools to ensure that the code-base meets all of the pre-defined testability requirements. However, for other NFRs, such as usability, Alpha relies heavily on manual verification and validation through user tests and by MEASURING EMOTIONAL RESPONSES e.g *" ... The best thing is to give people the ability to see something just for a few seconds. ... And then get them to provide a kind of like emotional response score"* (Manager). Some NFRs, such as usability, are not easy to validate through automated tests [56][55], especially in CSE environments. At Alpha, developers try to mitigate this challenge by defining and sharing development standards that describe how some NFRs are managed e.g For configurability, *"So I introduced a number of standards and built some packages to try and enforce those standards. So there's the standard for how we, on the server-side, get configuration from the environment"*. In addition, an organization like Alpha may benefit from capturing NFRs directly in source control [55] by using code and related artifacts to define rules that represent the metrics for an NFR. The process of putting an NFR directly in source control helps developers set an objective metric for an NFR and helps with sharing knowledge about what the organization defines as an acceptable threshold for the NFR, leading to an increase in shared understanding of the NFR.

Werner et al. describe the importance of the shared understanding of configuration management as configurability is an NFR that also encompasses process quality [55], and supports the rapid development, configuration and deployment of software products. Prior research has described comprehensive configuration management [31] for continuous integration and continuous delivery as a key component of CSE. Many organizations use more than a single configuration tool, such as Docker, to create a software environment and applications, as observed by Werner et al. [55] within the three organizations in their study. An organization would benefit from continuous integration and delivery incorporated in CSE practices when they invest in building a shared understanding of NFRs such as configurability. Shared understanding is enhanced through the practices recommended by Werner et al. [54], such as creating shared standards for configuration management and effective communication. At Alpha, developers use an in-house configuration tool for setting up their environment and application. Developers also use Process Street [44], Coda, and GitHub for documenting configuration standards and collaboration, as these tools allow the developers to coordinate their activities around configuration management.

### B. Mitigating the effects of remote interaction on shared understanding in CSE

Previous research analyzing surveys during the COVID-19 pandemic has suggested that effective communication is essential for a software team's productivity. Communication is often affected by changes in a team's culture and inefficient communication could pose a challenge to reaching milestones [43]. As CSE organizations rely on frequent feedback cycles, the quality of communication within software teams may suffer when team members work from a distance, i.e. work remotely [15]. However, our findings suggest that Alpha was able to mitigate some of the challenges in remote communication by using Gather, a virtual video conferencing tool that tries to simulate the face-to-face physical workspace *"Gather has made it possible for us as a remote team to have informal conversations. You literally just walk your little character up to someone else's little character, and you're talking to them"*. Gather, a collaboration tool, has not been heavily researched on collaborative work and, to the best of our knowledge, in collaboration in software development. Our findings bring evidence about Gather's value in creating spaces that approach in-person interaction that is conducive to shared understanding in software engineering. A 2020 study of a distributed design team working during COVID-19 lockdowns [39] extensively used Zoom, Slack, and Google docs, identified difficulties with establishing a team spirit in a distributed setting due to the limited one-view per participant experience with Zoom

meetings during group sketching and the difficulties with understanding the team member's progress. We believe that Gather has allowed Alpha to achieve the practices suggested by this study for mitigating the challenge of simulating a co-located design space and encouraging screen-sharing for remote collaboration.

A key enabler for shared understanding is the ability to ask questions and receive feedback, regardless of background experience, education, or other demographic factors [9], that we identified as asking question in our practices for building a shared understanding. Previous research suggests other enablers of shared understanding in remote teams such as visualizing information, online updates and trustworthiness [35]. We further discuss our insights about these enablers at Alpha. First, we observed that Alpha uses wireframing for interface designs and screen-sharing to enable the team to set and understand the expectations for NFRs. This observation supports prior research on building shared understanding for supporting remote teams through building shared mental models with visual representations [26]. This observation also corroborates prior research that suggests the existence of reference systems as an enabler for shared understanding [25]. Second, frequent communication ensures that there is a shared understanding of how team members feel and how the project progresses [35]. Online updates like daily check-ins on Slack or formal or informal communication on Gather shows the importance of multiple forms of communication media [33] [29] and tools such as Gather, Slack, and file sharing. Third, mutual trust enables team members to collaborate and share feedback openly, thus improving shared understanding within remote teams [35]. Previous research suggests that mutual trust is a prerequisite for implicit shared understanding [25]. Previous research also suggests that regular communication, including formal and informal communication, plays a vital role in creating relationships within remote teams and thus in building trust within the team, regardless of the difference in background, location or culture [28]. Alpha built trust within the team mainly through informal communication on Gather or Slack, where team members are open to seeking help from each other, as observed *"If someone sends me a quick Slack message, and I look at it ... I will go straight to their desk and start to engage them in conversation so that we can whiteboard it or work through the problem"*.

### *C. How can organizations be proactive in building a shared understanding of NFRs?*

Through observations and the practices we identified in our study, we bring empirical evidence that further adds to and enhances the work by Werner et al.[54] on the *recommended* practices for building shared understanding i.e., how team communication and shared development standards form the fundamental base for building a shared understanding. In addition, we have identified a new practice of building a shared understanding through experience. Our first practice, discussing problems and solutions, is part of utilizing communication to develop a shared understanding. Developers can also initiate the creation of a shared understanding by asking questions, which is another form of communication. Validating NFRs through feedback is a form of communication that can happen through the continuous pipeline or in a meeting; regardless of *how* that communication takes place, the emphasis is on the continuous validation of NFRs that can build a shared understanding. An organization can leverage wireframes for interface designs, a form of shared development standards, that may enhance the proliferation of a shared understanding. Finally, developers can deepen their understanding of NFRs through experience at their organization and how that particular organization values NFRs.

Werner et al. [54] observed that an organization builds a shared understanding of NFRs primarily due to a reactive response to accidental lack of shared understanding, related to various reasons – *triggers* – such as scope creep, rework, regulatory requirements, accumulating technical debt, needs of an important customer, or a disruption of service. In Table III, we show the triggers that emerged from our data analysis and how they correspond to the identified practices. In the practices we observed at Alpha, the *deepening the understanding of NFRs through experience* is an example of such a reactive response, or where our data does not indicate that the response was the result of any of these or other triggers.

However, we categorized four of the five practices we identified at Alpha as being *proactive* in many instances, even though they were *reactive* in a few instances. Proactive practices do not occur as a result of a *trigger*, while reactive practices use different *triggers* to identify a need for building a shared understanding of NFRs [54]. One practice in particular sheds more light on how Alpha adapted to working and building shared understanding in remote settings: *wireframing interface designs*. Prior research suggests that using virtual shared spaces that enable the team to visualize solutions and interact with visualizations encourages better communication and enables shared understanding [4]. Through Gather's virtual shared space and screen-sharing feature, Alpha used tools such as Figma, an online whiteboard for visual representations with elements like blocks and shapes, to support collaboration with team members, thereby enabling the team to develop a shared understanding more quickly, as observed in prior research [4].

We observed that a proactive effort to build a shared understanding may not require a formal process or documentation. At Alpha, *"There's a [guild_name] guild and we meet once every two weeks to sort of check-in on NFRs. It's what practices are we going to try and use? What's working, what's not working? It's been helpful for talking about how we build things and NFRs in specific areas of our jobs."* The loose agenda of these meetings is rather informal and lends to the creative discussions around NFRs. Our findings indicate that tools such as Gather offer a medium for these informal, yet important, conversations to occur. Developers at Alpha unanimously describe Gather as a method that has created a sense of being connected to each other, despite their remote setting. *"I think Gather town has really helped with that, it's just a place where you can feel like you're next to the people*

*that you're working with"*.

Our findings have practical implications to software organizations as they show that while there is still a considerable amount of shared understanding built through reactive measures, there is also a sizable amount that is a direct result of some proactive practices. One developer indicated that he found some NFRs as follows: *"as you're going down into more details, during something like a journey map, that's where you start bumping into the non-functional requirements that may not have been clearly identified"*. However, when it comes to such NFRs, discussions did not happen until much later, as one participant noted, *"that's when you really start the back and forth communication on how you're going to address them...So I think it's during that process of drilling down into the details on how a project is going to be run, that you start hitting them, and the NFRs"*. Despite the complexity and importance of NFRs [52] [2], previous research observes that software developers discover NFRs during or after the software product implementation [42]. Eliciting NFRs at this stage may pose a challenge that could affect the entire software development process or lead to delays in software [42] [52]; therefore, it is prudent for an organization to ensure the conversations about NFRs are occurring early enough to facilitate building a shared understanding. The definition of *when* is left for future work; however, given the effects of architectural decisions on NFRs (or lack thereof), a shared understanding should be continually built throughout the development cycle, perhaps allotting enough time for "just-in-time" engineering [18].

## VI. Threats to Validity

We acknowledge the limitations of our qualitative study through four components of the total quality framework [46]: credibility, analyzability, transparency and usefulness.

For the *credibility*, the selection of Alpha may suffer from sampling bias as we focused on an organization willing to partner with us; however, we verified that their practices align with CSE through observation by one of our researchers. In addition, we interviewed participants with various roles and levels of experience. To limit conscious or unconscious bias during data collection, we explained that study participation was completely anonymous and would not cause any consequence or risk to the participants. We started each interview session by exploring the definition of shared understanding. Afterwards, we explained shared understanding in the context of our research with examples to ensure that each participant had the same level of knowledge about shared understanding.

For *analyzability*, we used a transcription tool on the recorded audio from the interviews and one of the authors listened and verified each transcript. We used thematic analysis through the open, axial, and selective coding process from grounded theory [30]; we describe this process in Section III. We used the inter-rater agreement process [37] to align our codes and categories within our coding scheme. We used reflective memos during the coding process to describe patterns observed in the data collected for reference.

For *transparency*, we provide a replication package (https://doi.org/10.5281/zenodo.6273497) with our codebook, interview questions, and data about our inter-rater agreement sessions; however confidential data are excluded due to our non-disclosure agreement. Although we believe we reached saturation of codes in our data analysis, one limitation to our coding process may be the subjectivity of the exhaustiveness of our codes which may influence research results.

For *usefulness*, we do not propose that our results and findings hold true for all remote CSE organizations. We acknowledge that our single case study research provides results limited to our partner organization. However, to increase the usefulness of our findings, more case studies like ours would be helpful with a focus on how remote software organizations can proactively build and maintain a shared understanding of NFRs, and how to evaluate a shared understanding of NFRs.

## VII. Conclusions and Future Work

A shared understanding of NFRs is important for the success of many software projects. However, managing NFRs can pose a significant challenge to software organizations due to factors such as the conflicting nature of NFRs and the lack of shared understanding of NFRs. In addition, shared understanding of NFRs in CSE is still understudied and there is insufficient literature to show how organizations can effectively build a shared understanding.

While prior research has explored how NFRs are managed in CSE, considering the challenges and best practices for managing NFRs, there is still the question of *how* CSE organizations can build a shared understanding of NFRs. This paper describes the practices for building a shared understanding of NFRs, through a case study of a remote CSE organization. Through discussing the practices for building a shared understanding of NFRs in CSE, we observe that CSE organizations are proactively building a shared understanding, although still reactively building a shared understanding in several instances. Furthermore, we discuss the limitations to a shared understanding of NFRs in CSE, highlighting the similarities and differences between the limitations described in our study and prior research.

With respect to the practical implications of our study, we acknowledge the limitations to our research as a single-case study, however, we believe that our research adds valuable empirical evidence to this research area and is a useful starting point to further explore how CSE organizations can build a shared understanding of NFRs. In future research, we seek to evaluate the effectiveness of proactive practices for building a shared understanding of NFRs in CSE.

## Acknowledgments

We thank our partner organization and employees for their time and collaboration. We acknowledge Nowshin Nawar Arony and Neha Koulecar (University of Victoria) for their assistance in our study. Our research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

## References


[1] Par J Agerfalk, Brian Fitzgerald, Helena Holmstrom Olsson, Brian Lings, Bjorn Lundell, and Eoin Ó Conchúir. "A framework for considering opportunities and threats in distributed software development". In: *Proceedings of the International Workshop on Distributed Software Development, Paris* 29 (2005), pp. 47–61.

[2] David Ameller, Claudia Ayala, Jordi Cabot, and Xavier Franch. "How do software architects consider non-functional requirements: An exploratory study". In: *2012 20th IEEE International Requirements Engineering Conference (RE)*. IEEE. 2012, pp. 41–50.

[3] Paul Atkinson. *Ethnography: Principles in practice*. Routledge, 2007.

[4] Aruna D Balakrishnan, Susan R Fussell, and Sara Kiesler. "Do visualizations improve synchronous remote collaboration?" In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2008, pp. 1227–1236.

[5] *BDD Testing and Collaboration Tools for Teams*. URL: https://cucumber.io/.

[6] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[7] Alexander Bick, Adam Blandin, Karel Mertens, et al. *Work from home after the COVID-19 Outbreak*. 2020.

[8] Eva Alice Christiane Bittner and Jan Marco 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*. IEEE. 2013, pp. 106–114.

[9] Philip Cash, Elies A Dekoninck, and Saeema Ahmed-Kristensen. "Supporting the development of shared understanding in distributed design teams". In: *Journal of engineering design* 28.3 (2017), pp. 147–170.

[10] *Cloud storage for work and home - Google Drive*. URL: https://www.google.com/intl/en_ca/drive/.

[11] *Coda — The doc that brings it all together.* URL: https://coda.io/.

[12] Peter Darch, Annamaria Carusi, and Marina Jirotka. "Shared understanding of end-users' requirements in e-Science projects". In: *2009 5th IEEE International Conference on E-Science Workshops*. IEEE. 2009, pp. 125–128.

[13] *Datadog: Cloud Monitoring as a Service*. URL: https://www.datadoghq.com/.

[14] Adam Debbiche, Mikael Dienér, and Richard Berntsson Svensson. "Challenges when adopting continuous integration: A case study". In: *International Conference on Product-Focused Software Process Improvement*. Springer. 2014, pp. 17–32.

[15] Advait Deshpande, Helen Sharp, Leonor Barroca, and Peggy Gregory. "Remote working and collaboration in agile teams". In: *International Conference on Information Systems, ICIS* (2016).

[16] Siva Dorairaj, James Noble, and Petra Malik. "Effective communication in distributed Agile software development teams". In: *International Conference on Agile Software Development*. Springer. 2011, pp. 102–116.

[17] Omar Elazhary, Colin Werner, Ze Shi Li, Derek Lowlind, Neil A Ernst, and Margaret-Anne Storey. "Uncovering the benefits and challenges of continuous integration practices". In: *IEEE Transactions on Software Engineering* (2021). DOI: 10.1109/TSE.2021.3064953.

[18] Neil A. Ernst and Gail C. Murphy. "Case studies in just-in-time requirements analysis". In: *2012 Second IEEE International Workshop on Empirical Requirements Engineering (EmpiRE)*. ISSN: 2329-6356. Sept. 2012, pp. 25–32. DOI: 10.1109/EmpiRE.2012.6347678.

[19] Brian Fitzgerald and Klaas-Jan Stol. "Continuous software engineering: A roadmap and agenda". In: *Journal of Systems and Software* 123 (2017), pp. 176–189.

[20] Martin Fowler, Jim Highsmith, et al. "The agile manifesto". In: *Software development* 9.8 (2001), pp. 28–35.

[21] *Gather - A better way to meet online*. URL: https://www.gather.town/.

[22] Cristina B Gibson and Susan G Cohen. *Virtual teams that work: Creating conditions for virtual team effectiveness*. John Wiley & Sons, 2003.

[23] *GitHub: Where the world builds software . GitHub*. URL: https://github.com/.

[24] Martin Glinz. "On non-functional requirements". In: *15th IEEE International Requirements Engineering Conference (RE 2007)*. IEEE. 2007, pp. 21–26.

[25] Martin Glinz and Samuel A Fricker. "On shared understanding in software engineering: an essay". In: *Computer Science-Research and Development* 30.3 (2015), pp. 363–376.

[26] Gabriela Goldschmidt. "To see eye to eye: the role of visual representations in building shared mental models in design teams". In: *CoDesign* 3.1 (2007), pp. 43–50.

[27] Brett Heasman and Alex Gillespie. "Neurodivergent intersubjectivity: Distinctive features of how autistic people create shared understanding". In: *Autism* 23.4 (2019), pp. 910–921.

[28] Kaisa Henttonen and Kirsimarja Blomqvist. "Managing distance in a global virtual team: the evolution of trust through technology-mediated relational communication". In: *Strategic Change* 14.2 (2005), pp. 107–119.

[29] Pamela J Hinds and Suzanne P Weisband. "Knowledge sharing and shared understanding in virtual teams". In: *Virtual teams that work: Creating conditions for virtual team effectiveness* (2003), pp. 21–36.

[30] Rashina Hoda, James Noble, and Stuart Marshall. "Grounded theory for geeks". In: *Proceedings of the 18th Conference on Pattern Languages of Programs*. 2011, pp. 1–17.

[31] Jez Humble. "Continuous delivery sounds great, but will it work here?" In: *Communications of the ACM* 61.4 (2018), pp. 34–39.

[32] Jan Ole Johanssen, Anja Kleebaum, Barbara Paech, and Bernd Bruegge. "Practitioners' eye on continuous software engineering: An interview study". In: *Proceedings of the 2018 International Conference on Software and System Process*. 2018, pp. 41–50.

[33] Shawn Jordan and Robin Adams. "Perceptions of success in virtual cross-disciplinary design teams in large multinational corporations". In: *CoDesign* 12.3 (2016), pp. 185–203.

[34] Michelle E Kiger and Lara Varpio. "Thematic analysis of qualitative data: AMEE Guide No. 131". In: *Medical teacher* 42.8 (2020), pp. 846–854.

[35] Jonas Kniel and Alice Comi. "Riding the Same Wavelength: Designers' Perceptions of Shared Understanding in Remote Teams". In: *SAGE Open* 11.3 (2021), p. 21582440211040129.

[36] Robert E Kraut, Robert S Fish, Robert W Root, and Barbara L Chalfonte. "Informal communication in organizations: Form, function, and technology". In: *Human reactions to technology: Claremont symposium on applied social psychology*. 1990, pp. 145–199.

[37] J Richard Landis and Gary G Koch. "The measurement of observer agreement for categorical data". In: *Biometrics* (1977), pp. 159–174.

[38] Rakesh Kumar Lenka, Srikant Kumar, and Sunakshi Mamgain. "Behavior driven development: Tools and challenges". In: *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. IEEE. 2018, pp. 1032–1037.

[39] Tatiana Losev, Sarah Storteboom, Sheelagh Carpendale, and Søren Knudsen. "Distributed synchronous visualization design: Challenges and strategies". In: *2020 IEEE Workshop on Evaluation and Beyond-Methodological Approaches to Visualization (BELIV)*. IEEE. 2020, pp. 1–10.

[40] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. "An investigation into the notion of non-functional requirements". In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. 2010, pp. 311–317.

[41] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. "Managing conflicts among non-functional requirements". In: *Australian Workshop on Requirements Engineering*. University of Technology, Sydney. 2009.

[42] Abderrahman Matoussi and Régine Laleau. "A survey of non-functional requirements in software development process". In: *Technical report TR-LACL-2008-7, University of Paris-Est (Paris 12)*. LACL (Laboratory of Algorithms, Complexity and Logic). 2008.

[43] Courtney Miller, Paige Rodeghero, Margaret-Anne Storey, Denae Ford, and Thomas Zimmermann. ""How Was Your Weekend?" Software Development Teams Working From Home During COVID-19". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 624–636.

[44] *Process Street — Checklist, Workflow and SOP Software*. URL: https://www.process.st/.

[45] Sadhana Puntambekar. "Analyzing collaborative interactions: Divergence, shared understanding and construction of knowledge". In: *Computers & education* 47.3 (2006), pp. 332–351.

[46] Margaret R Roller and Paul J Lavrakas. *Applied qualitative research design: A total quality framework approach*. Guilford Publications, 2015.

[47] Ravi Sen and Sharad Borle. "Estimating the contextual risk of data breach: An empirical approach". In: *Journal of Management Information Systems* 32.2 (2015), pp. 314–341.

[48] Helen Sharp, Yvonne Dittrich, and Cleidson RB De Souza. "The role of ethnographic studies in empirical software engineering". In: *IEEE Transactions on Software Engineering* 42.8 (2016), pp. 786–804.

[49] *Slack is where the future works*. URL: https://slack.com/.

[50] Darja Smite, Nils Brede Moe, Georgiana Levinta, and Marcin Floryan. "Spotify guilds: how to succeed with knowledge sharing in large-scale agile organizations". In: *IEEE Software* 36.2 (2019), pp. 51–57.

[51] Peter Törlind and Andreas Larsson. "Supporting Informal Communication in Distributed Engineering Design Teams". In: *International CIRP Design Seminar: 16/05/2002-18/05/2002*. 2002.

[52] Saeed Ullah, Muzaffar Iqbal, and Aamir Mehmood Khan. "A survey on issues in non-functional requirements elicitation". In: *International Conference on Computer Networks and Information Technology*. IEEE. 2011, pp. 333–340.

[53] Colin Werner. "Towards A Theory of Shared Understanding of Non-Functional Requirements in Continuous Software Engineering". In: *2021 IEEE 29th International Requirements Engineering Conference (RE)*. IEEE. 2021, pp. 498–503.

[54] Colin Werner, Ze Shi Li, Neil Ernst, and Daniela 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)*. IEEE. 2020, pp. 90–101. DOI: 10.1109/RE48521.2020.00021.

[55] Colin Werner, Ze Shi Li, Derek Lowlind, Omar Elazhary, Neil A Ernst, and Daniela Damian. "Continuously managing NFRs: Opportunities and challenges in practice". In: *IEEE Transactions on Software Engineering* (2021).

[56] Liang Yu, Emil Alégroth, Panagiota Chatzipetrou, and Tony Gorschek. "Utilising CI environment for efficient and effective testing of NFRs". In: *Information and Software Technology* 117 (2020), p. 106199.