CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Discovering Error Handling Strategies for Improving End User Experience in Collaborative Distributed Real Time Systems

Master's thesis in Computer science and engineering

David Andreasson

Karl Gunnarsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Discovering Error Handling Strategies for Improving End User Experience in Collaborative Distributed Real Time Systems

David Andreasson
Karl Gunnarsson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Discovering Error Handling Strategies for Improving End User Experience In Collaborative Distributed Real Time Systems
David Andreasson
Karl Gunnarsson

Supervisor: Olof Torgersson, Computer Science and Engineering
Advisor: Staffan Olsson, Yolean
Examiner: Morten Fjeld, Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Screenshot from the design artifact

Typeset in LaTeX
Gothenburg, Sweden 2024

Discovering Error Handling Strategies for Improving End User Experience In Collaborative Distributed Real Time Systems
David Andreasson
Karl Gunnarsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis investigates error-handling strategies in collaborative distributed real-time systems, focusing on enhancing the end-user experience. Collaborative environments like Yolean, characterized by complex distributed architectures, often suffer from errors that negatively impact user satisfaction and system functionality. Through a combination of Research Through Design, Requirements Engineering, and Design Thinking, this study identifies error patterns and evaluates user preferences for managing these errors. Our research involved semi-structured interviews, affinity diagramming, storyboarding, prototyping, and iterative user testing to develop an understanding of user expectations and effective error-handling strategies. We developed and refined prototypes addressing two main scenarios: Distributed architecture and collaborative interactions with human-induced errors. The proposed error-handling strategies aim to provide intuitive and non-disruptive user support, enhancing system usability and reliability. The findings from the study are summarized as an initial framework to guide the exploration of error-handling strategies in such environments. Results from the study indicate that the design artifact improved the end-user experience compared to the current state of the Yolean application. However, further research would be needed to draw generalized conclusions and guide further refinement of the proposed framework.

Keywords: Error-handling, Error-presentation, Error-mitigation, Distributed-systems, Collaborative-system, Real-time-systems, Interaction-design

Acknowledgements

We extend our sincere gratitude to our supervisor, Olof Torgersson, for his guidance, support, and encouragement throughout the duration of this project. We would also like to thank our advisor, Staffan Olsson, and Senior Developer Anton Lindgren at Yolean, for their valuable input, expertise, and assistance in navigating the complexities of our project. Their perspectives provided invaluable insights that enriched our understanding and enhanced the quality of our work. Additionally, we express our heartfelt appreciation to our fellow students who participated in the user testing phase of our research. Lastly, we acknowledge the contributions of all individuals who supported us in various capacities, whether through discussions, feedback, or encouragement. Your support has been invaluable, and we are deeply grateful for your assistance throughout this endeavor.

David Andreasson, Karl Gunnarsson, Gothenburg, 2024-06-06

Contents

List of Figures

List of Tables

1

Introduction

Yolean is a web-based application that streamlines collaborative real-time planning, with the goal of reducing meeting times for its users [1]. Developed by the startup company Yolean, it has found significant utility in the construction sector, serving purposes in project planning and production management. To accommodate its growing user base and ensure scalability, Yolean is architectured as a distributed system, comprising containerized microservices orchestrated by Kubernetes to dynamically adjust resource allocation based on demand [2]–[5]. Real-time editing features are facilitated through the combined use of Apache Kafka, a distributed event streaming platform with persistent storage, and the WebSocket Protocol through a full duplex connection [6], [7].

Despite presenting itself as a monolithic system to clients, the underlying distributed nature of Yolean's architecture introduces complexities in error handling. These complexities stem from the asynchronous and decentralized nature of distributed systems, making error reasoning, handling, and presentation challenging. Consequently, Yolean has tasked us with the critical objective of discovering effective error-handling strategies. Despite the prominence of Human-Computer Interaction (HCI) research on handling human errors in digital systems, there remains a notable gap in understanding how errors from the systems should be managed and presented. Our project aims to investigate the following research questions:

- What are the possible error states of collaborative distributed real-time systems and how can these be identified?
- What error handling strategies exists, how could these be presented to the user, and how do different approaches affect the end-user experience and their goals?
- How can insights from the broader field of error handling in collaborative real-time distributed systems be unraveled through methodologies such as Research Through Design, Design Thinking, Requirements Engineering, and iterative user testing?

By exploring these questions, we seek to bridge this gap and develop effective error-handling strategies in the collaborative distributed real-time systems space. Through a combination of theoretical analysis and practical experimentation using Research Through Design, Design Thinking, and Requirements Engineering, we aim to enhance the usability and reliability of systems like Yolean, ultimately improving

the user experience.

1.1 Background

Setting the stage for the project, and adding some context to the research questions, we performed an initial analysis of possible error states in systems and an architectural analysis of the Yolean application.

1.1.1 Initial Error Awareness Matrix

After an initial discussion with our supervisor at Yolean, we recognized the need for an initial model for reasoning about error states within a distributed system like Yolean. To address this, we modeled an initial error awareness matrix, serving as a foundational tool for understanding and reasoning about the various combinations of error states between servers and clients in the system. The error awareness matrix represents a high-level abstraction of potential error scenarios, capturing different states of awareness for server and client components, which is visualized in figure 1.1. Each quadrant in the matrix corresponds to a unique combination of client and server awareness states: **Server Aware - Client Aware**, **Server Aware - Client Not Aware**, **Server Not Aware - Client Aware**, and **Server Not Aware - Client Not Aware**.

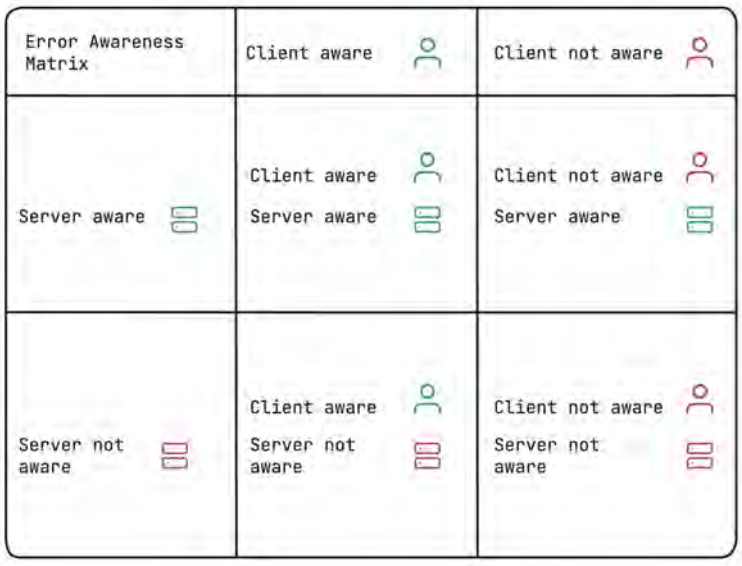

Figure 1.1: An initial high-level abstraction of a model for the different error states that might occur in a distributed real-time system.

In the Server Aware quadrants, the server component is aware of the error state, while in the Server Not Aware quadrants, the server is not aware of an error state. Similarly, the **Client Aware Quadrants** indicate that the client component is aware of the error state, while the **Client Not Aware Quadrants** indicate that it is not aware of an error state. By systematically categorizing potential error states

based on the awareness levels of both the client and server components, this matrix serves as an initial tool for reasoning about possible error-handling strategies.

1.1.2 Overview of Yolean Application Architecture

To further reason about possible errors that might occur in collaborative distributed real-time systems, we created a systems architecture diagram representing the Yolean architecture, as visualized in Figure 1.2.

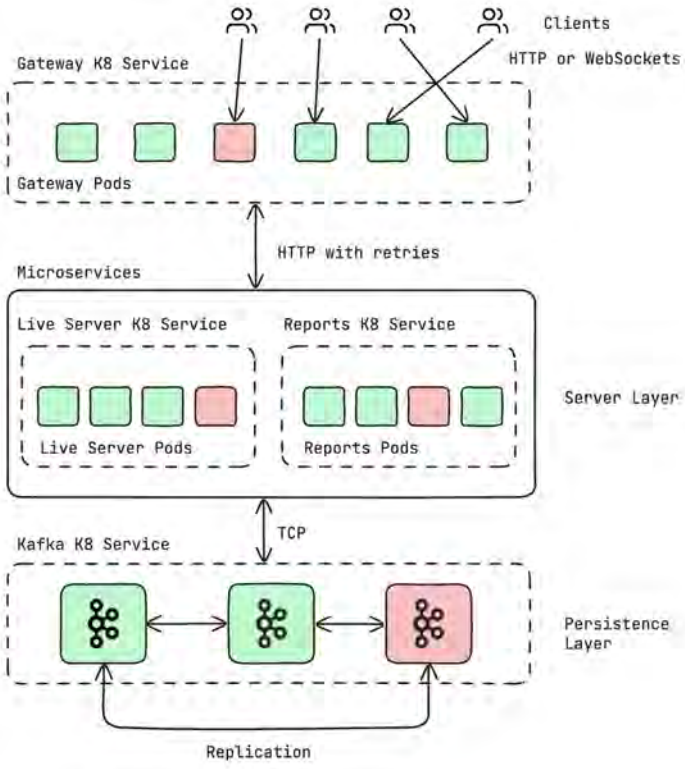

Figure 1.2: Yolean Application Architecture Diagram.

The Yolean architecture can be categorized into three main layers: The gateway layer, the server layer consisting of the different microservices, and the persistence layer for storing data in Apache Kafka. Each colored square in the diagram represents a Kubernetes pod, with green indicating healthy pods and red indicating unhealthy ones. The gateway layer serves as the initial point of entry for client requests, handling authentication and routing to other services in the Microservices layer. Multiple gateway pods are deployed for scalability to distribute the load on the system. In the microservices layer, various services are available for user interaction, such as the live server responsible for operations in the visual planner (e.g., moving meeting notes, adding meetings, adding tasks). Figure 1.2 also illustrates a separate microservice for generating reports from decisions, indicating its self-contained nature. Similar to the gateways, microservices are encapsulated as Kubernetes services, with multiple pods for redundancy and load distribution. The persistence layer is responsible for storing operations performed in the visual planner.

The architecture diagram visually demonstrates that errors can occur at multiple levels within a distributed real-time system, given the nature of pods being swapped out or ending up in an unhealthy state.

This overview of the Yolean application architecture provides insights into its structure and components, highlighting potential points of failure that follow with scalability considerations. Understanding these aspects is crucial for developing effective error-handling strategies in collaborative distributed real-time systems. The architecture diagram visually demonstrates that errors can occur at multiple levels within a distributed real-time system, given the nature of pods being swapped out or ending up in an unhealthy state. Leveraging insights from our analysis of the Yolean architecture, we gain another tool for reasoning about errors and possible error-handling strategies.

2

Theory

In this chapter, we delve into the theoretical underpinnings that form the foundation of our research on error-handling strategies in collaborative distributed real-time systems. We begin by exploring some related work related to error handling in HCI, Distributed Systems, and Real-Time Systems. We then turn to the fundamentals of distributed systems and examine the principles and challenges associated with designing and managing distributed architectures. Next, we turn our attention to collaborative real-time systems, investigating the unique characteristics and requirements of systems that facilitate real-time collaboration among multiple users. Building upon this understanding, we delve into designing for error, error classification, exploring different categories and types of errors that can occur in distributed real-time environments. Finally, we explore the crucial aspect of error presentation, considering various methods and approaches for effectively communicating errors to users to enhance the user experience and system usability. Through a comprehensive exploration of these theoretical concepts, we lay the groundwork for developing robust error-handling strategies tailored to the specific needs and challenges of collaborative distributed real-time systems.

2.1 Related Work

In this section, we explore existing literature and research relevant to error-handling strategies in collaborative distributed real-time systems. Understanding how errors previously have been addressed and managed in various contexts provides valuable insights and informs our approach to developing effective error-handling strategies.

2.1.1 Error Handling in HCI

The research on errors and error handling within the realm of HCI has primarily focused on slips and mistakes, slips meaning unintended actions taken by a user and mistakes meaning disallowed actions taken by a user. Furthermore, slips are unintentional errors caused by inattention, and mistakes are conscious errors caused by a misunderstanding between the user and the interface [8]. In addition to focusing on slips and mistakes in the realm of HCI, research has been conducted on the tone and wording of error messages and how these affect the user experience. One study compared, apologetic, non-apologetic, and neutral error messages and found that

users perceived the system with apologetic error messages as more usable and aesthetically appealing [9]. In addition, the apologetic system was the least frustrating system for users. Developers tend to be aware of the importance of usability when designing a system, However, software can often lack crucial elements that would render interfaces truly user-friendly, thereby failing to meet users' needs and expectations. A study with software developers focused on error message design found that developers tend to include information that is not only beneficial to the user but also to themselves such as error codes [10]. Yet, a majority of the developers acknowledge that it was not appropriate to do so. The result of this research consists of several guidelines that designers should follow when displaying errors to their users. For instance, error messages should be easy to understand and be noticeable to the user. The user should be able to locate the errors effortlessly and should be able to redeem or resolve the error [11].

2.1.2 Error Handling in Distributed Systems Utilizing Kubernetes

An evaluation of architecture for deploying microservice-based applications in Kubernetes clusters hosted in both public and private clouds [12] sheds light on key insights relevant to error handling in distributed systems:

- Kubernetes' repair actions may not be sufficient for providing high availability, particularly in scenarios such as node failures. This underscores the importance of robust error-handling mechanisms to complement Kubernetes' built-in capabilities.
- Adding redundancy emerges as a critical factor in reducing downtime and enhancing availability. The study underscores the significance of redundancy mechanisms in distributed systems to bolster fault tolerance and resilience.

While Kubernetes offers mechanisms for managing containerized microservices and integrating redundancy mechanisms like replica sets and load balancing, failures at the node and pod levels remain inevitable [12], [13]. However, research in effectively handling or mitigating the impact of these failures on end users remains sparse. This gap motivates further investigation into error-handling strategies that bridge the gap between system failures and end-user experiences.

2.1.3 Error Handling in Real-Time Systems

Error handling in real-time systems presents unique challenges, particularly in ensuring predictability and reliability amidst potential exceptions and disruptions. Hard real-time systems, characterized by strict timing constraints and deterministic behavior, pose additional complexities in error management. One paper in this domain reflects on handling exceptions in Hard real-time environments, identifying it as a less elaborated topic within the field [14]. The paper underscores the necessity of preventing exceptions whenever possible and handling them consistently and safely when they do occur. This approach aligns with the user experience goal of minimizing disruptions and providing clear guidance to users in error scenarios. While Yolean

might not be classified as a Hard real-time system, proactive error prevention and robust error-handling strategies remain relevant. By proactively addressing potential sources of errors and implementing robust error-handling strategies, the project aims to enhance the overall user experience of collaborative distributed real-time systems. The concept of layer-by-layer predictability emphasizes the importance of consistency and reliability across all levels of system design [14], a principle briefly introduced in the Yolean Architecture section. Similarly, in user experience design, maintaining consistency in interface behavior and responsiveness is paramount for ensuring a seamless user experience. By adopting a layered approach to error handling and system design, the project strives to uphold predictability and reliability at every stage of user interaction, ultimately contributing to an improved user experience quality.

2.2 Distributed Systems

Distributed systems represent a paradigm shift from traditional centralized architectures, offering adaptive, user-customizable graphical interfaces and enabling collaboration and information sharing across geographically dispersed user communities [15]. Distributed systems exhibit a set of intriguing properties. Multiple functional units enable dynamic resource allocation to independent execution threads. Typically, both physical and logical units, are distributed yet interconnected via networks, with operating systems managing and orchestrating their components. Each functional unit may operate with its local operating system instance, communicating with others for synchronization and ensuring consistency among dependent threads. In distributed systems, cooperative autonomy refers to the ability of individual functional units to collaborate and coordinate their actions towards a common goal, even though they operate autonomously. Additionally, distributed systems exhibit independence during partial failures, meaning that if one component fails, it does not necessarily disrupt the entire system. Instead, the system should handle failures gracefully and continue functioning as much as possible.

Despite their advantages, distributed systems also present notable challenges. Performance and reliability depend heavily on the underlying network, with dependencies among various components and functional units adding complexity [15]. Furthermore, a distributed system can impede productivity when unfamiliar machines crash unexpectedly. Network management becomes imperative to address the complexities of network support, with limitations potentially leading to communication bottlenecks and information exchange constraints.

2.2.1 Transparency in Distributed Systems

Transparency is a fundamental principle in the design of distributed systems, aiming to create a seamless user experience where accessing local and remote resources is indistinguishable [16]. The goal is to ensure that the behavior of the distributed system mirrors that of a hypothetical centralized system, erasing any distinction between the two. A hypothetical centralized system may serve as the foundation when

designing distributed systems. Therefore, transparency guides the construction of distributed systems, emphasizing the need for consistency and equivalence with their centralized counterparts. Thus, transparency refers to the ability to conceal the intricacies of the distributed system's architecture from its users. Furthermore, transparency in distributed systems emerges across various levels:

- **Location Transparency:** Location transparency ensures that users can access resources without knowing their specific network location [15]. It involves using a simple, location-independent name to access resources, regardless of their location. Three levels of location transparency are defined:
  - Level 0 involves networking applications enabling communication between computing systems [15].
  - Level 1 hides the presence of multiple interconnected systems from the user [15].
  - Level 2 extends this to network-wide services for shared resource usage, such as shared file systems, further concealing the underlying infrastructure [15].
- **Access Transparency:** Access transparency ensures uniform access to local and remote resources [15]. An example is the World Wide Web, where Uniform Resource Locators (URLs) enable transparent access to files stored locally or remotely.
- **Device Transparency:** Device transparency refers to transparent access to locally or remotely located devices, ensuring consistent user experience regardless of device location [15].
- **Replication Transparency:** Replication transparency ensures seamless access to replicated resources, even when users are unaware of their replication status [15].
- **Failure Transparency:** Failure transparency masks link failures or node crashes [15]. For instance, ensuring uninterrupted access to replicated files even if an individual file server crashes.
- **Concurrency Transparency:** Concurrency transparency embeds synchronization control into distributed system implementations, facilitating parallel and concurrent access to shared resources across dispersed users [15].
- **Migration Transparency:** Migration transparency allows resources to migrate between nodes without affecting running applications [15].
- **Execution Transparency:** Execution transparency permits processes to run on different runtime systems [15].
- **Performance Transparency:** Performance transparency allows dynamic system reconfiguration to optimize performance based on changing load characteristics [15].

- **Scalability Transparency:** Scalability transparency supports system and application extensions without structural modifications [15].
- **Language Transparency:** Language transparency enables interaction between components implemented in different programming languages, ensuring seamless communication regardless of language variations [15].

2.2.2 CAP Theorem

In distributed system design, a common problem is the daunting task of balancing multiple conflicting objectives. The CAP theorem encapsulates one of the fundamental challenges in distributed system design [17]. Formulated by computer scientist Eric Brewer in the early 2000s, the CAP theorem highlights the inherent trade-offs between three crucial properties of distributed systems: consistency, availability, and partition tolerance. The CAP theorem stipulates that a Web service must make trade-offs when pursuing consistency, availability, and partition tolerance, sacrificing one of the attributes to achieve the two others.

Delving further into the CAP theorem, it is essential to dissect each of its three components: consistency, availability, and partition tolerance [18]. Consistency, the first aspect of the CAP theorem, refers to ensuring that each server delivers the correct response for every request, aligning with the intended service specifications. Availability, the second aspect of the CAP theorem, signifies that every request ultimately garners a response, emphasizing the importance of timely replies. Even demanding an eventual response can pose challenges within CAP's framework, acknowledging that delayed responses may be as detrimental as no response in practical systems. Partition tolerance, the third aspect, concerns the system's resilience to communication breakdowns. To clarify, even if certain parts of the network become unreachable or isolated, the system should still be able to function to some extent. While each aspect contributes to the resilience and fault tolerance of the system, according to the CAP theorem, there are inherent trade-offs between them that architects must carefully navigate in distributed system design.

2.2.3 Microservices Architecture

Microservices Architecture is a modern approach to building distributed systems that emphasizes agility, scalability, and efficient resource utilization [19]. Unlike traditional monolithic architectures, Microservices Architecture breaks down applications into independent services of smaller scope and responsibility. This decentralization enables faster software delivery, easier maintenance, and dynamic scalability.

Key features of the microservices architecture include:

- **Decentralized Integration:** Services communicate directly rather than through a central bus, enhancing flexibility and decoupling [19].
- **Service Independence:** Each microservice operates independently, promoting functional separation, reusability, and scalability [19].

- **Dynamic Scalability:** Services can be scaled independently based on demand, optimizing resource usage and responsiveness [19].
- **Lightweight Communication:** Microservices communicate via lightweight protocols such as REST APIs, simplifying communication and reducing overhead [19].

However, Microservices Architecture also presents challenges:

- **Increase Network Complexity:** Managing network traffic and ensuring reliability become more complex as the amount of microservices grows [19].
- **Service Coordination:** Coordinating interactions between microservices requires careful planning and implementation of discovery mechanisms and messaging frameworks [19].
- **Data Management:** Dividing monolithic databases poses challenges in data consistency and management, requiring strategies for partitioning and synchronization [19].

In conclusion, Microservices Architecture offers a promising approach to building scalable and resilient distributed systems. By embracing decentralization and service independence, organizations can accelerate software delivery and adapt to evolving user needs. However, addressing challenges in network complexity, service coordination, and data management is essential for successful adoption.

2.3 Collaborative Real-time Systems

In the development of real-time collaborative systems, the advantage of standard web technologies can be utilized due to their widespread support across modern devices, facilitating seamless collaboration through built-in web browsers and wireless network capabilities [20]. Web-based systems offer unparalleled accessibility, requiring only URL access through browsers, thereby eliminating the need for installing native applications on user devices and enhancing user convenience. However, employing web technologies for collaborative real-time systems presents challenges, particularly in communication channels. While newer protocols like Web Socket enable bidirectional communication, older browsers may require alternative methods such as long polling, impacting system performance and compatibility. Nevertheless, given Yolean's utilization of Web Sockets, our focus narrows to newer browsers supporting advanced protocols.

Addressing conflict errors in web-based collaborative systems is a significant challenge, with the TWICE framework proposing client-side handling strategies [20]. By utilizing the system clock of the providing server, the TWICE tool kit establishes a shared time base among clients, enabling consensus on event ordering. Conflicts, manifested as unordered events, are identified and managed based on event type. TWICE suggests locking strategies for non-undoable events, delaying processing until conflict-free execution is assured while advocating for the rollback of undoable events. Discarded events, such as cursor movements, are ignored if superseded by

newer information. The TWICE framework's custom distributed event handling strategy addresses the limitations of cloud-based solutions like Firebase, which rely on internet connectivity and may not be viable in offline or local network scenarios [21]. While the TWICE toolkit was designed for low-capability or low-infrastructure environments, it also offers adaptability by enabling integration with infrastructure-supported services through event mechanism replacement.

2.4 Designing for Error

When designing for errors, the end goal is to handle the errors as gracefully as possible [22]. One can employ multiple strategies for handling errors. The first strategy is to produce a system that minimizes or avoids errors. The second strategy is to enable users to deal with errors suitably, thus failing gracefully. Lastly, the system could provide users with contextual information about the intended action, to further improve the understanding of the implications of said action. The uppermost level of categorization of errors consists of two major categories: mistakes and slips. These categories differ in the level of intention, where if the intentions are not appropriate, the error is categorized as a mistake, and if the action is not aligned with the user's intention, the error is categorized as a slip [8], [22]. While it might not be feasible to build systems that eliminate errors, one can employ strategies to mitigate errors significantly:

Avoiding Error Through Appropriate Representation: The choice of representation in system design profoundly impacts the occurrence of errors [22]. For instance, when files are represented as typed strings of characters, errors can arise from misremembered or misspelled file names or working in the wrong directory. On the other hand, when files are represented with icons on the screen and specifying them by clicking on them, this illustrates how changing the representational format can mitigate errors. In this scenario, it becomes impossible to specify a nonexistent file, and spelling errors are eliminated. On the other hand, each representational format introduces its own set of trade-offs, underscoring the complexity of these design decisions.

Avoiding False Understandings: False understandings of system properties can lead to errors, highlighting the importance of providing comprehensive information to users [22]. Often, individuals generalize from single experiences, which can enhance learning efficiency but also result in inaccurate conclusions. An anecdote from a student further illustrates this concept: initially struggling to delete files on a UNIX system, the student observed someone using the "rm" command and inferred it as the standard delete command, which in this case was correct. However, observing another student exiting a UNIX-based mail system, using the "x" command, he once again concluded that this was the standard command for exiting the session. This assumption proved false when he encountered discrepancies in the UNIX mail system. Despite deleting messages with the "x" command, they persisted in subsequent sessions due to misunderstanding the command's function, in which an administrator clarified that "x" was an abnormal exit command, not intended for terminating the mail session.

Minimizing Errors That Result From Slips: Three key categories of slips are mode errors, capture errors, and description errors, which can be used as a source for specific areas for improving the minimization of errors [22]. Mode errors, where users perform actions appropriate for one mode while in another, underscore the need for clear feedback on system states. Although modes are inevitable in complex systems, efforts should focus on minimizing them and providing distinct mode indicators. Description errors occur due to insufficient specification of actions, leading to ambiguity and erroneous acts. Consistency in the command structure is crucial to prevent users from deriving actions incorrectly based on analogies with similar aspects of the device. Capture errors arise from overlap in sequences for infrequent and frequent actions, often resulting in performing the more common action. Minimizing overlapping sequences and providing feedback at critical decision points can mitigate this error class.

2.4.1 Error Classification

Two initial categories for error classification are First Time Errors and Consecutive Errors [23]. First Time Errors refer to errors that occur the first time when a user interacts with an input form. Since users encounter these obstacles for the first time, they have no prior experience to guide them, leading to mistakes. On the other hand, Consecutive Errors occur after the input or action is validated once. Thus, users who encounter error messages that indicate incorrect input still submit the data incorrectly in consecutive tries. Consecutive errors highlight persistent issues or misunderstandings that may persist even after providing feedback.

Further error classification is integral for deepening the understanding of the reasons behind errors, their occurrence, and how they should be addressed and communicated to users. While error classification in Human-Computer Interaction (HCI) is not extensively studied, it holds significant importance, especially in critical domains such as power plants and medical systems [24]. For our thesis, we will adopt a subset of the CSNI Taxonomy, a classification system developed for reporting erroneous events in nuclear power plants. This taxonomy consists of 24 categories, with 12 focusing on human factors.

The selected subset of categories we will utilize includes:

Specified task not performed due to omission of act: The system enters an erroneous state because the user misses or forgets part of the action they are trying to perform [24].

Commission of erroneous act: The system enters an erroneous state as the user performs a disallowed action [24].

Commission of extraneous act: External factors affecting the system put it into an erroneous state [24].

Sneak-path, accidental timing of several events or faults: The system enters an erroneous state due to coincident or co-effect with something else [24].

By employing these categories, we aim to systematically analyze errors within our context and gain deeper insights into their nature and origins.

2.4.2 Error Presentation

Dealing with error messages is often viewed as a necessary evil of software interaction, and experts widely advocate for error prevention as the ideal approach [25], [26]. However, recognizing that errors inevitably occur, it becomes paramount for users to notice and comprehend their causes. To optimize the effectiveness of error messages, adherence to the three following key guidelines is essential:

Visibility

Error messages should be prominently displayed and positioned close to the source of the error, ensuring users can effortlessly identify the problem's origin and minimize cognitive strain [27]. Techniques such as bold text, high-contrast formatting, and using red styling for indicators should be used to enhance visibility. It is crucial to consider accessibility by employing multiple indicators instead of relying solely on visual cues.

Communication

Effective communication is vital in error messages, necessitating clear and easily understandable language to assist users in recovering from an error state [27]. Messages should be concise, avoiding vague or generic content. Maintaining a positive and supportive tone is essential, as the primary goal is to offer constructive guidance rather than assign blame to the user.

Efficiency

Error messages should explain the issue and prioritize saving the user's time and effort [27]. They should provide actionable instructions for problem resolution, facilitating a better understanding to prevent future occurrences. To mitigate common mistakes, offer feedback when errors occur. Allow users to amend their original action instead of starting anew and provide suggestions for alternative actions. When focusing on visibility, communication, and efficiency, error messages can become more user-friendly and constructive.

Moreover, users transitioning between different mental modes and the timing of error presentation also hold significant implications for user experience [23]. The authors elaborate on the timing aspect by stating that their results indicate that immediately presenting errors upon occurrence neither significantly harms nor necessarily aids user experience. The authors also present two mental modes in conjunction with the timing aspect: Completion Mode and Revision Mode. During Completion Mode, users primarily focus on completing their task and often overlook errors until they transition into Revision Mode. The study does not determine whether the users actively ignore the error feedback or do not notice it during Completion Mode. Developers should optimally embed error messages within the interface and the input forms, allowing errors to be displayed individually or collectively. Alternatively, if errors are presented in modals, it is suggested to display them individually.

Demanding users to memorize multiple problems can incur frustration and a higher rate of consecutive errors.

Lastly, Lewis and Norman elaborate upon how the systems should respond in general and introduce the following goals: Determining the user's intention and signaling when inappropriate actions occur or are about to occur [22]. After determining a user's intention, the following strategies are proposed:

Forcing Function

One effective response is to implement a forcing function, which halts further action until the problem is corrected [22]. For example, consider a car that won't move unless the user starts the engine, an example of an inherent forcing function. Forcing functions ensures self-detection of errors without the need for explicit error messages.

Gag

Another approach is the "gag" technique, where errors are dealt with by preventing the user from continuing, thereby blocking impossible intentions [22]. For instance, systems may lock the keyboard upon encountering errors, effectively gagging further input until the issue is resolved.

Warn

Warnings inform users of potentially dangerous situations but allow them to proceed if they choose [22]. In some interfaces, unavailable actions are shaded to signal their illegality without forcing the user to comply.

Do Nothing

Some systems opt for a "do nothing" approach, where attempting an illegal action yields no visible response [22]. This allows users to infer that their action was unsuccessful without disrupting their workflow with explicit error messages.

Self Correct

Systems may attempt to guess a legal action based on the user's input, such as auto-correct features in text editors [22]. However, automatic corrections must be accompanied by robust undo functionalities to mitigate potential errors.

Let's Talk About It

Initiating a dialogue with the user upon encountering problems represents a step towards true interaction with the system [22]. For example, Lisp systems often provide detailed error messages and entry into a debugging environment, enabling users to explore and resolve issues collaboratively with the system.

Teach Me

In the "teach me" approach, the system queries the user to understand ambiguous commands or phrases [22]. This iterative process of clarification and learning enhances the system's ability to interpret user input accurately over time.

2.4.3 Error Detection and Recovery

Furthermore, detecting errors promptly is crucial for initiating effective recovery and minimizing wasted time and effort for the user [22]. Early detection is particularly vital to prevent errors from accumulating and complicating diagnosis, especially for new users who may struggle to differentiate between correct and incorrect steps. Slips, where the performed action differs from the intended one, are generally easier to detect than mistakes, where the intention itself is flawed. However, even slips can be challenging to uncover due to the discrepancy between the level at which actions occur and the level at which users form intentions. Cognitive hysteresis further complicates error detection, as individuals may persist with a decision despite evidence of its incorrectness. Overcoming this bias requires systems to provide sufficient information for effective diagnosis and counteracting cognitive hysteresis, emphasizing the importance of clear and immediate error indication as a key design objective.

Correcting errors in system design involves implementing mechanisms that facilitate error recovery and provide users with the means to rectify mistakes effectively [22]. Incorporating features like "undo" facilities emerges as a crucial aspect of error correction, enabling users to revert to previous states and mitigate the impact of errors. However, the implementation of such features poses challenges, requiring consideration of technical and conceptual complexities. Additionally, a system should provide users with essential information to aid in error recovery, including information for understanding the current system state, tracing back to previous states, and knowing available alternatives [28]. Exploring strategies such as initiating dialogues with users, seeking clarification, and employing automatic correction mechanisms like "Self Correct" can further enhance error correction processes [22]. Overall, effective error correction in system design necessitates a multifaceted approach that addresses technical challenges while prioritizing user understanding and empowerment in navigating and resolving errors.

2.4.4 Challenges in Designing for Error

Two common challenges when designing for error is the problem of level and failure in detecting problems [22]:

The Problem of Level

A significant challenge in designing error messages lies in effectively communicating the nature of the problem to users [22]. Often, despite the system detecting an error and the user recognizing that an error occurred, there remains considerable difficulty in pinpointing the exact nature of the error. This difficulty stems from the existence of multiple levels of intentions. For instance, consider the scenario of encountering a "longjmp botch" error while formatting files on a terminal. The error message, displayed at a low level of program execution, fails to align with the user's high-level intention of formatting a file, leading to confusion and ambiguity regarding the source of the error.

In manual operations, detecting errors step by step is relatively straightforward, but

in automated systems, identifying the precise point of failure can be challenging [22]. Users often spend significant time fixing the wrong aspects of a program or task due to a lack of information about the level at which the problem occurred. Understanding user intentions and levels can aid in tailoring responses to specific contexts, but it does not guarantee error resolution, as evidenced by various cases where knowledge of intentions fails to help. Consider the example of an individual attempting to unlock a car door. Initially, the individual assumes the problem lies with the key insertion, then progresses through various hypotheses until realizing the attempt is made on the wrong car. Such examples highlight how users tend to address problems at the lowest level before gradually exploring higher levels of action and intention.

Addressing the problem of levels requires presenting error messages at appropriate abstraction levels and providing users with tools to explore errors comprehensively [22]. The "Let's Talk About It" strategy proposes presenting messages at the highest level of abstraction, indicating the presence and seriousness of a problem while empowering users to delve into deeper levels to understand the root cause. By facilitating constructive interaction and exploration, systems can potentially overcome the challenge of conveying error information effectively across different levels of user understanding and system operation.

Failure to Detect Problems

Often, actions may achieve a legal effect without being detected as errors by the system, yet they fail to advance the user towards their goal or align with their intentions [22]. Users' ability to detect such states is often unreliable, as demonstrated in various scenarios, including word-processing tasks where learners may rationalize significant errors to fit their perception of what should be happening. Despite seemingly adequate evidence, serious failures may go undetected, prompting Norman to propose three hypotheses to explain these difficulties: relevance bias, partial explanation, and overlap between model and world.

- **Relevance bias:** People seek confirmatory evidence over disconfirming evidence, favoring information that aligns with their hypotheses, even if incomplete or irrelevant [22].
- **Partial explanation:** Errors go unnoticed as individuals accept partial agreement between expectations and observations, disregarding conflicting information [22].
- **Overlap of model and world:** Alignment between one's model of the world and reality hinders error detection, as deviations may go unrecognized due to rough agreement between the two [22].

2.5 Ethics

When conducting user research, it is imperative to prioritize research ethics, which involves ensuring that the rights, well-being, and dignity of participants are upheld [29]. The following guidelines are pertinent to this project:

- **Guidance documents:** These provide essential information on obtaining consent, drafting comprehensive consent forms and information sheets, and offer guidance on conducting research with specific user groups or on particular topics.
- **Standardized consent forms and information sheets:** These documents maintain consistency in language while allowing researchers to input study-specific details, ensuring clarity and transparency for participants.
- **Data policies for user:** Implementing a tailored data policy for user research is essential. This policy should comply with relevant data protection laws, outlining what personal data is collected, how it is stored, and the procedures for its handling.

Furthermore, when crafting user experiences, human factors and the product's future usage must be carefully considered. It is crucial to refrain from imposing personal biases or preferences onto the product and to avoid making excessive assumptions about future users [30].

3

Methods

In this chapter, we outline the methodological approach employed in our research to investigate and address the challenges of error handling in collaborative distributed real-time systems. We adopt a multifaceted methodology consisting of Research Through Design, Requirements Engineering, Design Thinking, and various design methods to guide our design and solution development process. Research Through Design serves as the cornerstone of our approach, allowing us to iteratively explore, prototype, and evaluate error-handling strategies within the context of real-world design scenarios. Complementing this, Requirements Engineering provides a systematic framework for capturing, analyzing, and prioritizing the needs and constraints of end-users and stakeholders, ensuring that our solutions align with their expectations and objectives. Furthermore, we leverage Design Thinking principles to foster empathy, creativity, and collaboration throughout the design process, enabling us to generate innovative solutions that resonate with the user's needs and preferences. Lastly, we draw upon a diverse array of design methods. By synthesizing these methodologies, we aim to develop effective and user-centric error-handling strategies that enhance the usability, reliability, and overall user experience of collaborative distributed real-time systems.

3.1 Research Through Design

Research Through Design (RTD) is a research method that focuses on how interaction designers can approach wicked problems. The method stresses the importance of design artifacts and how they should be used as an outcome for generating new knowledge [31]. RTD is an iterative method where each cycle consists of designing, prototyping, testing, and refining. RTD allows us to gain continuous insights and a deeper understanding of the problem space. To evaluate our research findings, we will use the following criteria: process, invention, relevance, and extensibility.

Process

The methods used should be described in sufficient detail such that the conducted research could be replicated. Researchers should provide a rationale for the method selection and documentation that allows for reproducibility.

Invention

The research findings should be novel and articulate the value of the findings.

Relevance

The findings should emphasize their applicability in real-world scenarios and highlight the advantages over the current state. Additionally, they should be extensible, allowing for further development and leveraging of the gained knowledge in future work.

3.2 Requirements Engineering

Requirements Engineering (RE) is a systematic process that involves eliciting, documenting, analyzing, and managing the requirements of a system. RE and HCI share many concepts and techniques, such as user-centered and participatory design. However, RE differs from HCI in terms of the scope of the design aspect. While RE often follows a more linear approach of specifying, designing, and implementing, it may overlook key facets emphasized in HCI, such as prototyping and gathering user feedback [32]. In our project, we will incorporate several steps from the RE process, including the following:

Scoping

Scoping in requirements analysis involves defining clear boundaries and the intended scope of a system, which might be challenging due to vague statements provided by clients and a lack of clarity about the system's purpose [32]. It is an iterative process that achieves clarity as stakeholders develop a deeper understanding of the domain.

Fact gathering

For fact gathering, data collection techniques are often used from systems analysis methodologies, which include interviews, observation, questionnaires, and literature review [32]..

Analysis

The analysis primarily focuses on goal decomposition by answering the '5W' questions: what is the purpose of the system, what objects are involved, where is the system located, when should things happen, and why is the system needed [32].

Modelling The output from the analysis step is processed using techniques drawn from structured system development methods and conceptual modeling. Informal modeling notations, including data flow diagrams and entity relationship diagrams [32].

Validation

Validation involves ensuring that users comprehend the implications of a requirements specification and agree that it accurately reflects their desires. In our case, we can perform validation through iterative user testing with our prototypes [32].

Trade-off analysis

Specifications often cannot fully satisfy identified requirements. Thus, Trade-off analysis becomes crucial when dealing with conflicting views from different stake-

holders. This activity involves comparing, prioritizing, and deciding between requirements or design options. Techniques like ranked lists or matrix-based methods, including decision tables, are valuable for this analysis [32]..

3.3 Design Thinking

Design thinking is a problem-solving methodology based on a user-centered design process. The design process strongly emphasizes understanding and empathizing with the end-users, defining problems, generating creative ideas, and iteratively testing and refining solutions [33]. There are several different models for design thinking, and in this thesis, we will use the Hasso-Plattner Institute model [34]. The model contains the following 6 phases.

Understand

The first phase's primary function is to create an understanding of the problem at hand by making general assumptions and researching the topic [34]. The goal is to create a collective perspective and identify potential knowledge gaps.

Observe

Observations and interviews aid in understanding and gathering insights about user patterns, emotions, and needs [34].

Define the point of view

The results from the previous steps are synthesized to identify promising insight to determine the direction of the solution development [34].

Ideate

During ideation, both individual and collaborative approaches are used for ideation to generate multiple solutions using a variety of methods. The focus is not to find the correct solution but to generate several potential ideas [34].

Prototype

Prototyping involves creating tangible representations of ideas generated during the ideation phase [34]. These prototypes facilitate communication and enable researchers to test the feasibility of their ideas.

Test

Prototypes are tested iteratively with relevant potential users, collecting feedback to assess core functions, technical feasibility, and usability [34]. Each testing iteration provides new insights and allows for improvements and changes.

3.4 Design Methods

In this subsection, we delve into methodologies and approaches that we potentially intend to employ in the design phase of our research. Through a blend of established

design principles and innovative approaches, we strive to develop robust and user-centered solutions that enhance the usability, reliability, and overall user experience of the systems under investigation.

3.4.1 Interviews

Interviews constitute a fundamental method for survey research, complementing techniques like questionnaires [35]. Through direct communication between researchers and participants, interviews offer a dynamic platform for gathering rich qualitative data, including experiences, opinions, attitudes, and perceptions. Depending on the research context, interviews may take various forms, such as structured, semi-structured, or unstructured, allowing researchers to tailor the approach to suit their specific objectives.

3.4.2 Questionnaires

Questionnaires are the other fundamental method for conducting survey research, facilitating the systematic collection of self-reported information from participants across various domains, including characteristics, thoughts, feelings, perceptions, behaviors, and attitudes [35]. Questionnaires allow researchers to gather data from a large sample of participants in a standardized manner, enabling efficient analysis and comparison. While questionnaires primarily rely on participants' self-reports, researchers may also complement questionnaire data with observational techniques. Observation can provide valuable insights by validating or augmenting self-reported information through direct observation of behaviors, environmental factors, or contextual cues.

3.4.3 Prototyping

Prototyping is commonly used in the design process to create tangible representations of design concepts and ideas [35]. These prototypes serve as artifacts that facilitate communication and collaboration within design teams and foster further interaction with stakeholders, clients, and users. The primary purpose of prototyping is to translate research findings and design concepts into tangible form, allowing for exploration, evaluation, and refinement of ideas. Prototypes vary in fidelity, representing different levels of detail and functionality.

Low-fidelity prototypes are characterized by their simplicity and often abstract nature. These prototypes commonly take the shapes of paper prototypes or digital wireframes [35]. These prototypes are highly useful during the early stages of the design process, facilitating rapid ideation and exploration of multiple design alternatives. Low-fidelity conveys basic design concepts and interactions, allowing designers to gather initial feedback and iterate on ideas efficiently.

In contrast, high-fidelity prototypes exhibit greater detail and realism, resembling the final product in appearance and functionality [35]. These prototypes are typically developed during the later stages of the design process and are instrumental during

evaluation and user testing. High-fidelity prototypes enable stakeholders and users to interact with a more refined version of the design, providing valuable insights into usability, aesthetics, and overall user experience.

3.4.4 Wizard of Oz

The Wizard of Oz technique is a valuable method utilized in the early stages of prototyping and user testing to simulate system interactions and gather user feedback [35]. In this approach, researchers play the role of "wizards," orchestrating system responses behind the scenes to create the illusion of a fully functioning system for users. The primary objective of employing the Wizard of Oz technique is to observe and evaluate user behaviors, preferences, and reactions to different system functionalities or prototypes. Researchers carefully orchestrate user interactions, ensuring users perceive the system as operational and engage with it naturally. By simulating system responses and maintaining the illusion of functionality, researchers can obtain valuable insights into user interactions and system usability, informing the iterative design and development process.

3.4.5 Critiques

Critiques are a pivotal element in the design process, serving as a mechanism for soliciting constructive feedback on design concepts from teachers, clients, stakeholders, or peers [35]. Critiques often utilize the following steps: Statements of meaning, where participants express what they found meaningful about the design. Designer questions, where the designer asks questions to the participants. Neutral questions, where the participants ask questions to the designer. Finally, permissioned opinions, where the participants may express their opinions if the designer has consented to hear them.

3.4.6 Affinity Diagrams

Affinity diagramming is a process that helps design teams organize and analyze research data effectively [35]. It involves capturing insights, observations, or requirements on individual sticky notes, clustering them based on their affinity or similarity, and forming research-based themes.

3.4.7 A/B Testing

A/B testing is an optimization technique that compares two versions of a design to determine which one performs better against a specific goal [35]. This method involves randomly assigning users to different versions (A and B), running tests until a statistically relevant sample size is reached, and analyzing the results to identify the more effective design.

3.4.8 Storyboards

Storyboards are an efficient tool for contextualizing technology within a visual narrative, enhancing both understanding and empathy for the user. By depicting scenarios, storyboards capture the interplay of societal, environmental, and technical factors that influence the interaction with a product [35]. They catalyze empathy-building early in the design process and facilitate the exploration of design alternatives. Utilizing storyboards at the outset of the design process fosters a deeper understanding of user needs and encourages innovative problem-solving.

3.5 Time Plan

This section outlines our project's preliminary roadmap, aligning with the frameworks and methods introduced in the Methodology section

3.5.1 Requirements Gathering

In the initial stage of the project, we focus on gathering requirements to gain a comprehensive understanding of the domain. Our primary objective is to gain a deep comprehension of the problem. To achieve this goal, we conduct interviews (4.4.1) with relevant parties, research literature, and gather data from Yolean. This phase encompasses fact gathering (4.2), understanding, and observation (4.3).

3.5.2 Scoping & Analysis

With all the knowledge, information, and data collected from the first step, we intend to analyze and process the data using methods like Mind mapping (4.4.3) and Affinity diagrams (4.4.8). The end goal is a well-defined problem and a suitable scope for the project but also being able to create personas (4.4.4) and begin transitioning into ideation for potential solutions that can be iteratively tested and evaluated in the following step. This part relates to defining the point of view (4.3) and analysis (4.2).

3.5.3 Low-fidelity Prototyping Iteration Loop

To facilitate testing of potential solutions and identifying problems early in the design process, the first iteration loop will focus on creating low-fidelity prototypes. Through ideation and prototyping, we intend to turn our solutions into testable components and use them in a Wizard of Oz-based prototype (4.4.6). This part relates to ideate and prototype (4.3).

3.5.4 User Testing and Evaluation

Using the low-fidelity prototype, we aim to identify the most promising ideas for solutions that should be further iterated upon and then transitioned into a final high-fidelity prototype. We will accomplish this through user tests (4.4.9) and presenting

the prototype to the stakeholders at Yolean to receive critique (4.4.7) and early input on what works and what does not work. This part relates to test (4.3) and validation (4.2).

3.5.5 High-fidelity prototyping iteration loop

Using the knowledge from the earlier sections of our process, we will transition into developing a digital high-fidelity prototype implemented in the Yolean application.

3.5.6 Final User Testing and Evaluation

We will incorporate the final high-fidelity prototype into a final larger-scale user test with both users and stakeholders, and we will use the data collected to try to answer our research questions.

4

Execution

In this chapter, we outline our process when designing and developing our design artifact, covering requirements gathering, scoping & analysis, low fidelity prototyping, initial user testing and evaluation, high fidelity prototyping, and final user testing and evaluation.

4.1 Background Meeting

The design process commenced with a meeting primarily consisting of our advisor and stakeholders from Yolean. This meeting served as a foundational step to contextualize the error states within the Yolean application. Collaboratively, we developed an initial error awareness matrix, as depicted in Figure 1.1, utilizing the method of design thinking. The objective was to deepen our understanding of the underlying issues and initiate scoping activities aligned with requirements engineering principles [32], [34]. While errors acknowledged by both the server and client could be addressed through robust code, our focus shifted towards scenarios where either the server or client remained unaware of errors, presenting intricate challenges in enhancing the end-user experience.

4.2 Literature Review

To gain a deeper understanding of the problem space, we conducted a comprehensive literature review focusing on error-handling strategies for distributed collaborative real-time systems. The objective of this review was twofold: first, to acquire a thorough understanding of the technical intricacies of distributed architectures, and second, to identify best practices and frameworks for designing effective error-handling strategies. The insights gained from the literature review were instrumental in guiding the design and development of our design artifact throughout the research-through-design process. The findings of the literature review are presented in the Theory chapter of the thesis.

4.3 Interviews

After establishing a sound theoretical foundation, we conducted semi-structured interviews to delve deeper into the problem space and gain insights into error han-

dling in distributed collaborative real-time systems. The interviews were conducted with typical users of systems like Yolean, including developers at Yolean and fellow students, with a total of (n=7) participants.

We asked participants the following questions to explore their experiences with similar systems and gather their opinions on error handling:

- Do you have any examples of errors you have encountered while using collaborative systems, such as Google Drive, Overleaf, and Figma?
- How did you respond or handle the problem that arose?
- Do you have any opinions on how you would have preferred the system to handle such errors?
- How would you like a system to communicate about errors visually in the interface?

Additionally, we presented participants with two scenarios from the Yolean application: one involving an unsynced state caused by a server or client error, and another focusing on errors arising from collaborative work on a task board. Furthermore, we asked the particpants to share their preferred methods for interface communication in handling and effectively communicating these error states.

Throughout the interviews, we aimed to capture diverse perspectives and insights that could inform the design and development of error-handling strategies in our research. The responses provided valuable insights into user preferences and expectations regarding error communication and resolution.

4.4 Affinity Diagramming

Using the notes from the interviews, we compiled the interview answers into a FigJam board to facilitate the analysis of qualitative data. Following the interviews, affinity diagramming was conducted to categorize and analyze the qualitative data (see Appendix A, Figure A.1). Several key categories emerged from this analysis:

- Problems
- Feelings and attitudes
- Current approaches
- Wishes
- Preferences for handling errors when the server or client is not aware of errors due to an unsynced state
- Preferences for handling errors caused by human factors during collaborative work

4.5 Mapping

After completing the affinity diagramming process and mapping the clustered data to theoretical concepts from the literature review (see Appendix A, Figure A.2), we utilized this mapped data to inform subsequent design decisions. Specifically, we leveraged the insights gained from this mapping exercise to guide the creation of storyboards and ideation sessions.

By aligning the reported problems, feelings, attitudes, current approaches, wishes, and preferences for error handling in Yolean with theoretical frameworks, we aimed to generate rich narratives and innovative design solutions that address user needs and theoretical underpinnings. This approach ensured that our design process remained grounded in empirical evidence and theoretical insights, providing a solid foundation for developing our design artifact.

4.6 Critique Session

After mapping the data into the theoretical concepts surrounding errors in distributed systems and designing for errors, we brought the mapped data into a critique session. By leveraging the stakeholders' expert domain knowledge of Yolean, two key areas emerged where we could explore error-handling strategies. Errors related to the distributed architecture of the system commonly align with either a commission of an extraneous act or a sneak path, where errors occur due to the accidental timing of several events or faults. On the other hand, errors occurring surrounding the human factor of working collaboratively fall under the category of failure to detect problems, avoiding error through appropriate representation, and avoiding false understandings, as the application state remains intact but the result does not align with the user's goals [22], [24].

During the critique session, discussions centered on how to address the problem of level for end users in distributed error scenarios and how the system could better respond when entering error states. Two distinct cases emerged for exploration in error handling strategy improvement. On the one hand, for the distributed aspect, a common occurrence for actual users of Yolean is working with the application with slow internet, which mirrors the behavior of the application if a live server pod were to be replaced during runtime. On the other hand, regarding real-time collaboration, a common use case involves collaboratively re-planning a process, where delays could introduce a cascade behavior of new tasks that need to be re-planned. Using the cases that emerged from the critique sessions as a foundation, we proceeded to create storyboards to further explore potential improvements in error-handling strategies.

4.7 Storyboarding

Following the discussions from the critique sessions, we created two storyboards to visually depict the identified scenarios from the critique session. They are presented in figure 4.1 and 4.2 below:

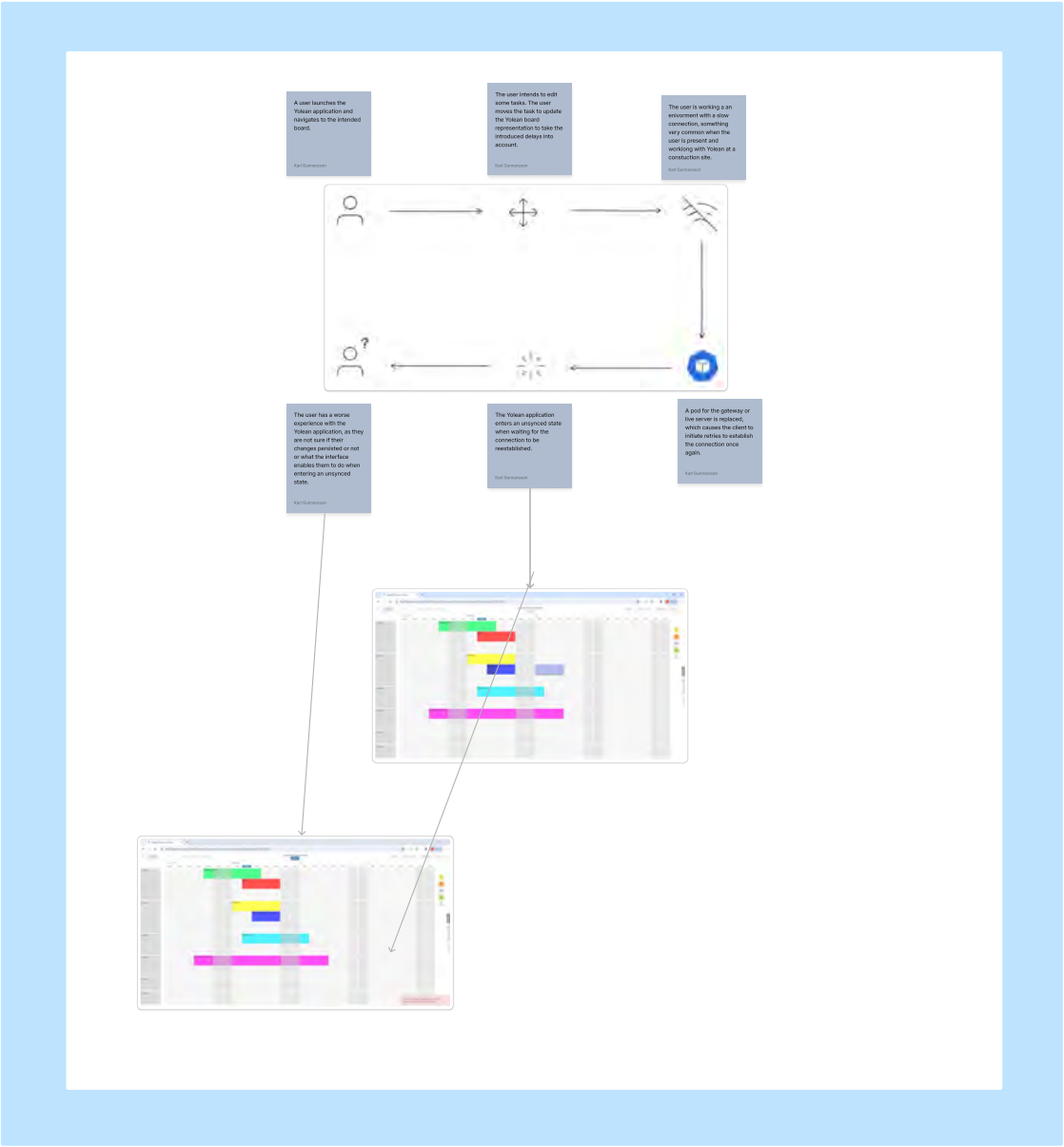

Figure 4.1: Storyboard of user flow trough the yolean application with an unstable distributed system.

In the distributed case storyboard, the steps are as follows:

- **User Launches Yolean Application**: The user opens the Yolean application and navigates to the desired board.
- **Intent to Edit Tasks**: The user intends to edit some tasks on the board.
- **Task Update Delay**: The user attempts to move a task, prompting the Yolean board representation to update. However, due to delays, this process takes longer than expected.

- **Slow Connection Environment**: The user is operating in an environment with a slow internet connection, which is common at construction sites.
- **Gateway or Server Pod Replacement**: A pod for the gateway or live server is replaced, causing the client to initiate retries to establish the connection.
- **Unsynced State**: As a result of the connection issues, the Yolean application enters an unsynced state, waiting for the connection to be re-established.
- **User Experience Impact**: The user's experience with the Yolean application worsens because they are unsure if their changes have been saved or not. They also lack clarity on what actions are available when the application enters an unsynced state.

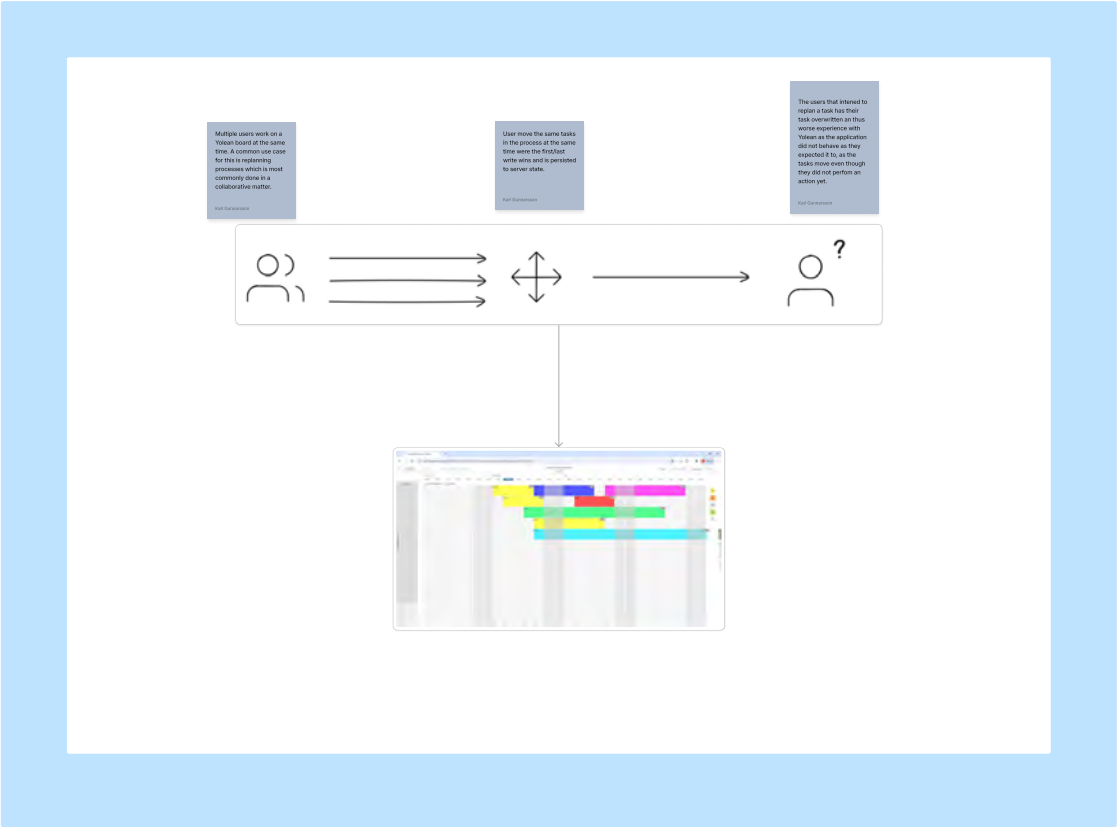

Figure 4.2: Storyboard of the user flow during collaborative work.

In the collaborative case storyboard, the steps are as follows:

- **Multiple Users Collaborating**: Several users are simultaneously working on a Yolean board, often to replan processes collaboratively.
- **Simultaneous Task Movement**: Users attempt to move the same tasks within the process simultaneously.

- **First/Last Write Wins Policy**: Yolean implements a policy where the first or last write operation takes precedence and is persisted to the server state.

- **Task Overwriting**: Due to the first/last write wins policy, if multiple users attempt to replan the same task, the changes made by one user overwrite those made by another user.

- **User Experience Impact**: Users intending to replan a task experience frustration and confusion because their changes are overwritten without warning. This results in a worse experience with Yolean as the application does not behave as expected, with tasks moving unexpectedly even though the user has not performed any action yet.

4.8 Ideation

Within the scope of the two storyboards, we began ideation for the design artifact. We used the mapping as a foundation for the brainstorming sessions to analyze and classify the behavior of the Yolean application and then introduced alternative approaches as sticky notes in our FigJam board [22]. The current behavior for handling errors in the distributed case uses a combination of a gag and forcing function approach, as a task can not be further moved after losing connection with the live server, which requires a full page reload to re-sync the state of the application, acting as the forcing function. Figure 4.3 visualizes the result of the brainstorming session for ideating on other approaches to handling errors when losing connection with server pods.

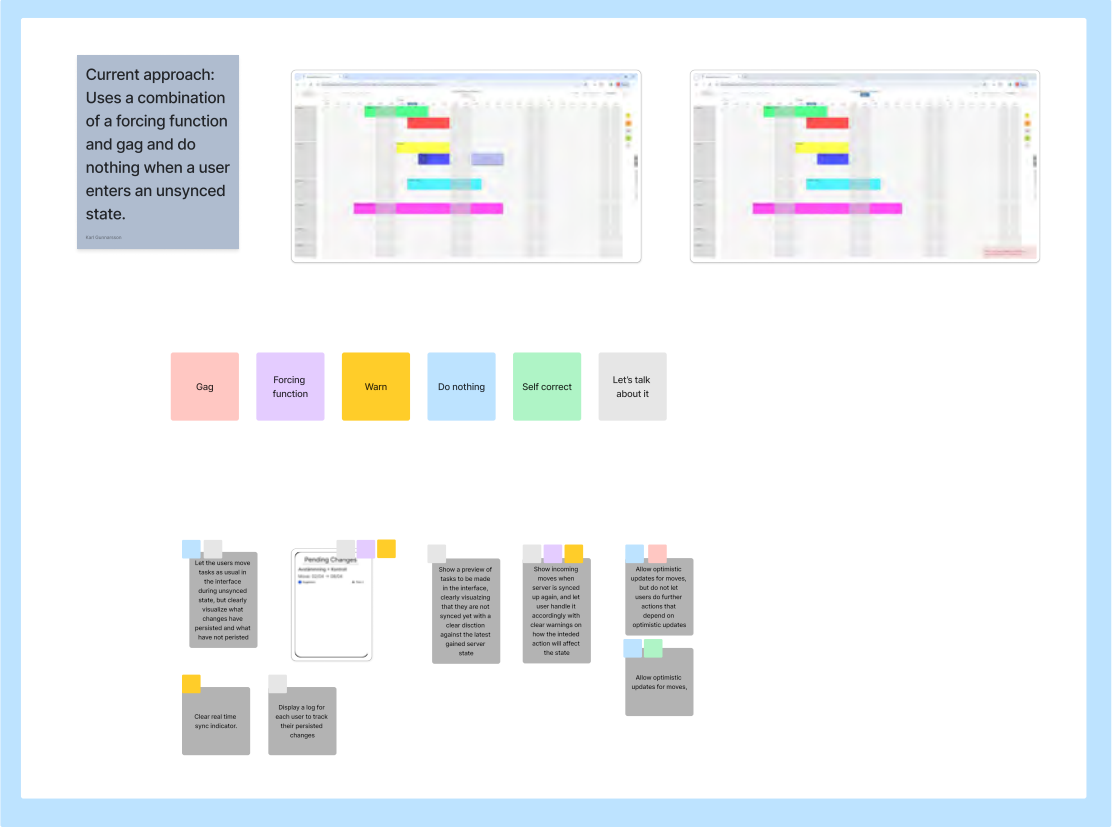

Figure 4.3: Result of the brainstorming session on error handling during unstable network conditions.

In the brainstorming sessions, we explored alternative approaches focusing on contrasting strategies such as warn, do nothing, self-correct, and let's talk about it.

For the collaborative case, the application behavior uses a combination of do nothing and self-correct, as no system state error occurs rather than that the state change is persisted to Kafka using the specific logic in the server side resolver for the user action, which most commonly is last write wins and is persisted. Figure 4.4 visualizes the result of the brainstorming session for ideating on other approaches to avoiding human factor errors when collaborating when re-planning task boards.

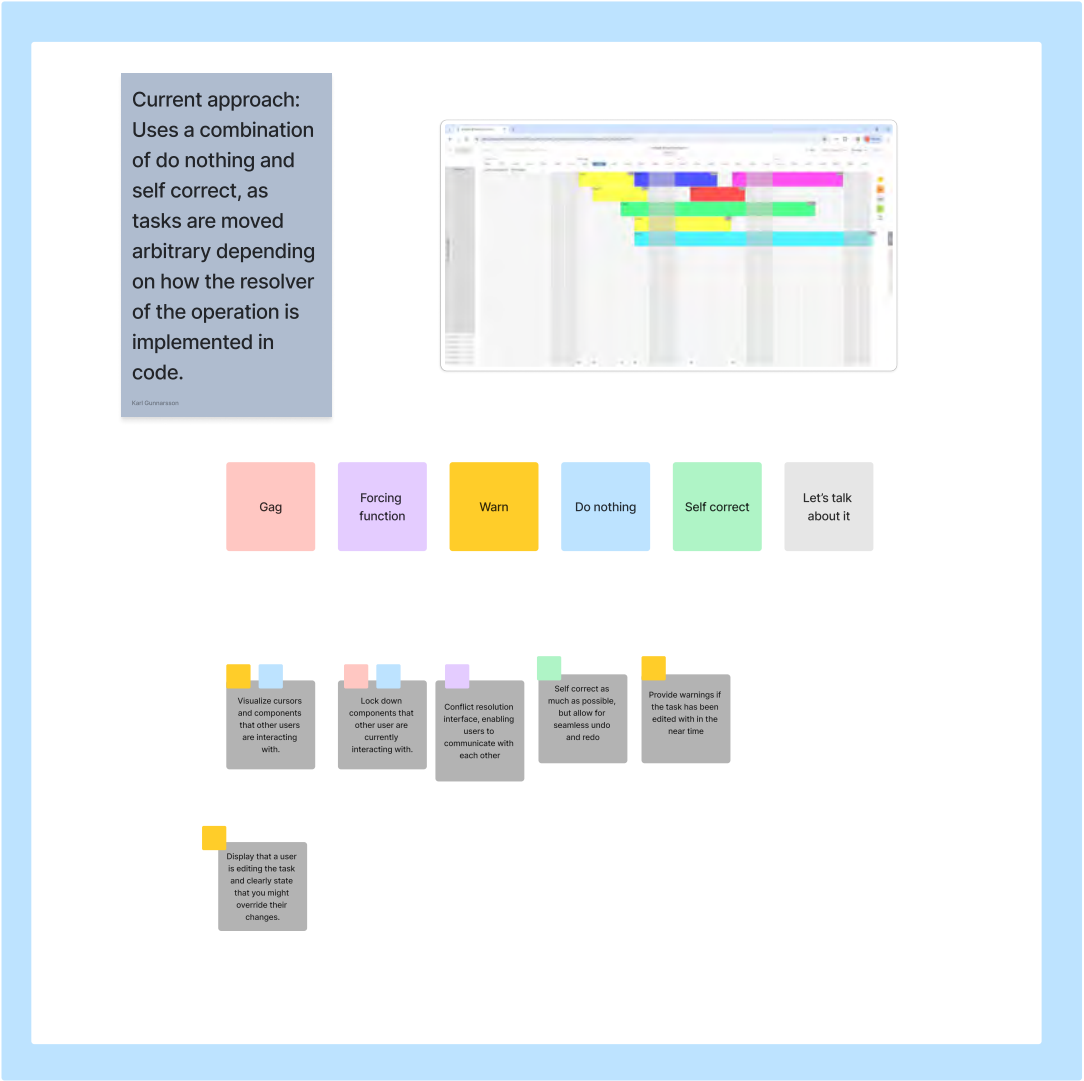

Figure 4.4: Result of the brainstorming session on error handling during collaborative work.

Alternative approaches were explored, covering warn, gag, do nothing, and self-correct.

4.9 Low Fidelity Prototyping

To progress from the ideation phase of design thinking to the prototyping phase, we developed two initial prototypes based on the insights gained from the brainstorming sessions. These prototypes aimed to explore new approaches for handling errors in both distributed and collaborative cases. Using the outcomes of our brainstorming sessions as a foundation, we created low-fidelity prototypes in Figma. These proto-

types served as representations of both the current state of the Yolean application and the explored changes. The prototypes were designed to test the effectiveness of the proposed solutions and gather feedback from stakeholders and potential users. By transitioning from ideation to prototyping, we aimed to iterate rapidly on our ideas and refine them further based on user insights.

4.9.1 Distributed Systems Case

Figure 4.5 visualizes the current behavior of the Yolean application if a live server pod were to be replaced:

- **Current behaviour for the Yolean application** The current behavior renders a preview of the task that a user has moved. The preview is visualized as a task with lower opacity and includes a loading spinner. The task cannot be further interacted with, effectively gagging the further moves or corrections for the user.

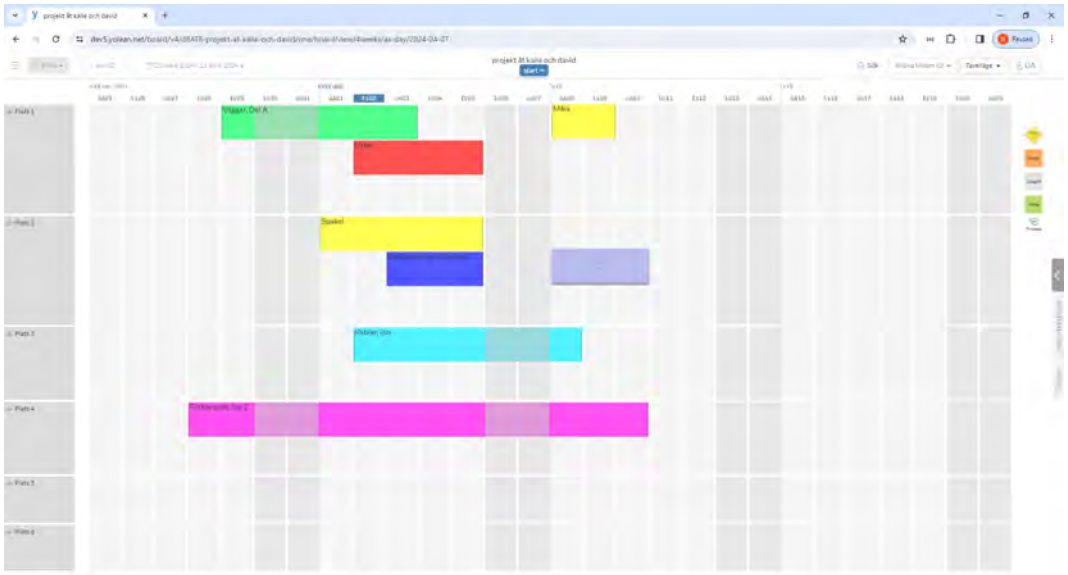

Figure 4.5: Screenshot of current pending changes solution in Yolean.

- **Pending changes visualisation**: The first prototype for the distributed case introduces a new modal that communicates the application state to the user, visualizing each move as an entry with the status of the action, the old and the updated date for the move, the task name, the task location, and the task trade. The first prototype is visualized in Figure 4.6 below:

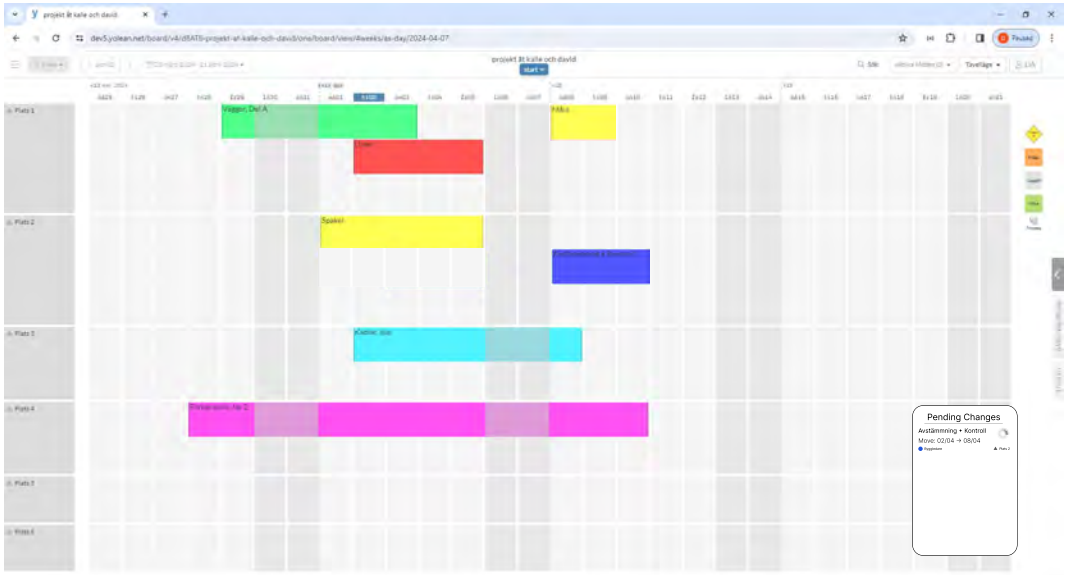

Figure 4.6: Low-fidelity prototype of pending changes visualisation.

This approach aligns more with a warning and let's talk about it guidelines to align the user's mental model of how the application changes state due to the distributed architecture of the Yolean application. Our goal of pursuing another approach was to further alleviate the problem of level by communicating the application state at a higher abstraction level for the end user [22].

- **Optimistic rendering**: In the second prototype, we removed the preview, and the task directly re-render on the client onto the new date. We kept the loading spinner to communicate the state of the action. The prototype also lets the user continue to interact with the task before the state has persisted in Kafka through the server, which is visualized in Figure 4.7 below:

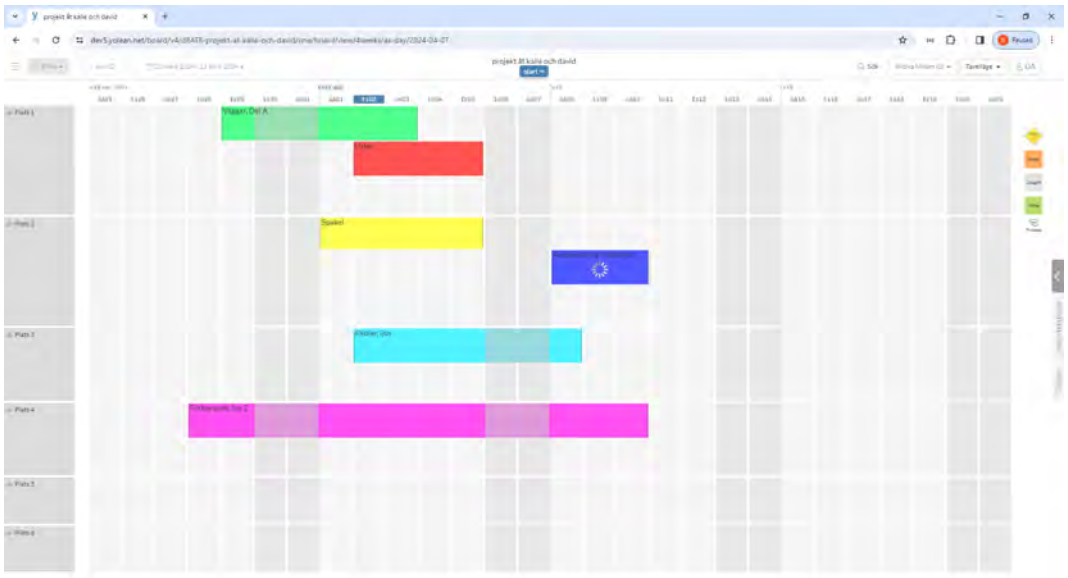

Figure 4.7: Low-fidelity prototype of the optimistic rendering visualisation.

This approach aligns with a do-nothing and self-correct approach, as the interface no longer prohibits the users from doing any actions while being in an error state, and thus relies on optimistic rendering on the client side instead of previews [22].

4.9.2 Collaborative Real Time Systems Case

Figure 4.8 visualizes the current behavior of the Yolean application when multiple users are interacting with the application:

- **Current behaviour for the Yolean application**: In the current state of the application, no indication exists to communicate that other users are interacting with the application.

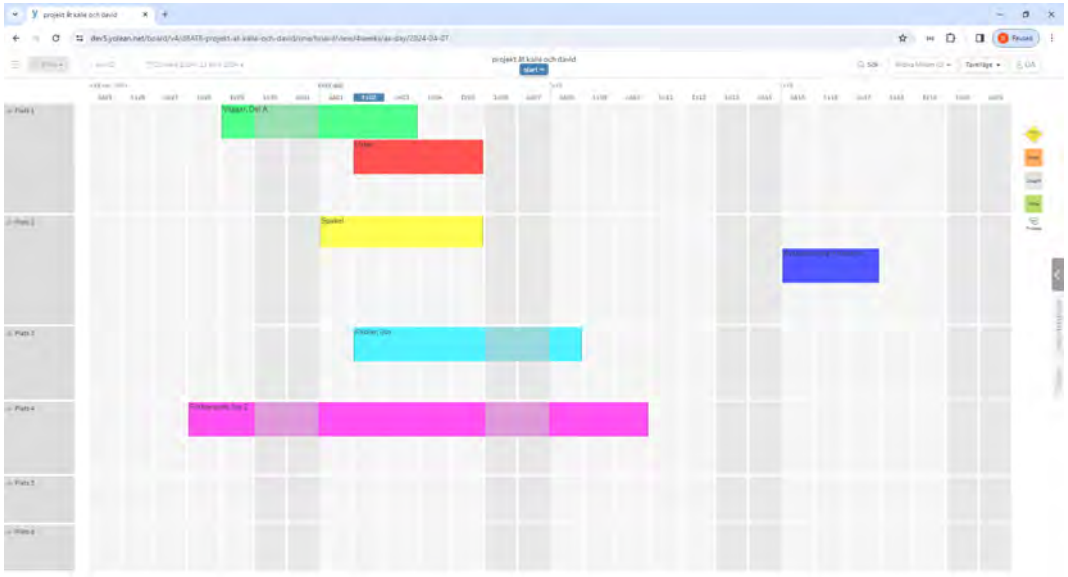

Figure 4.8: Current view of a Yolean task board.

- **Outline moved tasks**: For the first prototype, we introduced user cursors to visualize other users working in the interface, as well as outlining tasks when a user is interacting with it, which is visualized in Figure 4.9 below:

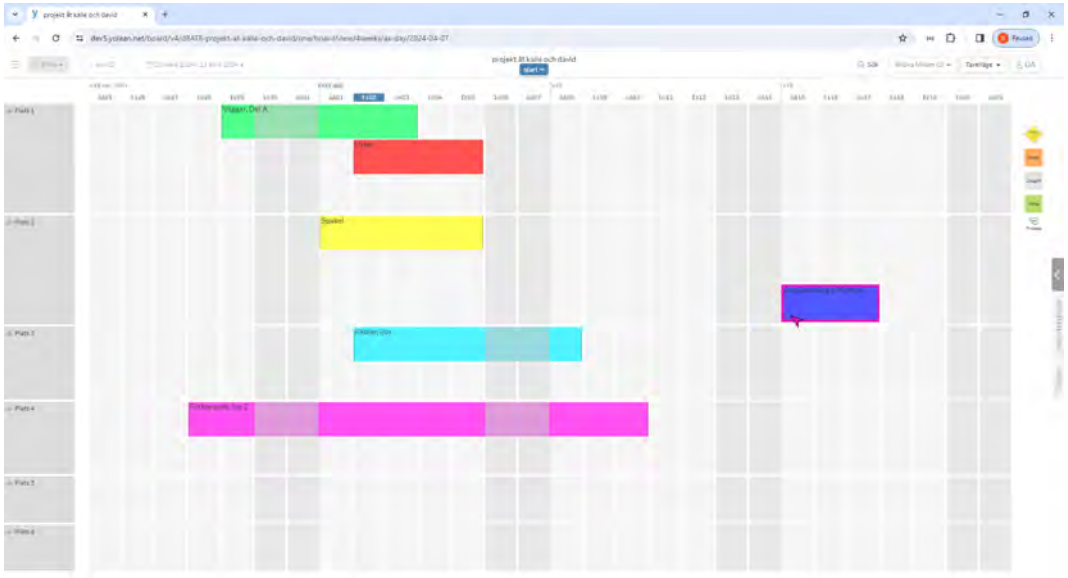

Figure 4.9: Low-fidelity prototype of the outline moved task visualisation.

The new visualizations introduce a combination of warn and do nothing behavior, as users now have access to an indication of what potentially might happen when interacting with tasks collaboratively with other users, but does not affect the actual behavior of user actions in the application, to avoid false understandings and the problem of failing to detect problems [22].

- **Lockdown**: The second prototype kept the visualizations from the first prototype, but also introduced a lock on tasks that other users are already working on, indicated by lowering the opacity of a task as well as an outline in the user cursor color, which is visualized in Figure 4.10 below:

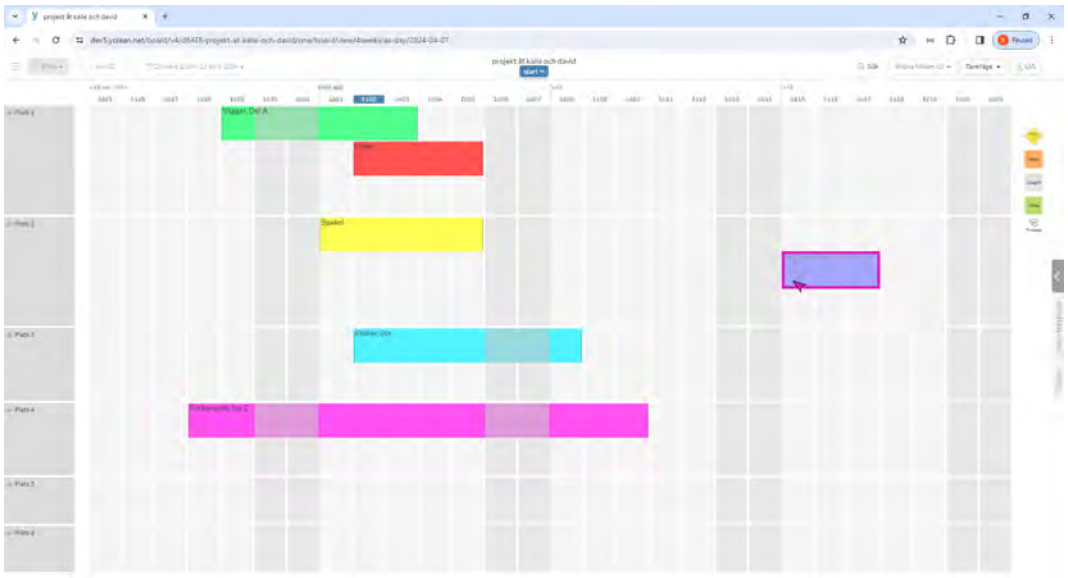

Figure 4.10: Low-fidelity prototype of the lockdown task visualisation.

The lockdown prototype combines a gag and do-nothing approach, as trying to interact with the locked task will not yield any results for the end user [22].

4.10 Initial User Testing

After creating the low-fidelity prototypes, we transitioned to the first iteration of the test phase in the design thinking process. The user test consisted of (n = 9) participants. The participants were to test the current state and the two new prototypes for the distributed and collaborative cases modeled through the storyboards. The user interacted with the prototypes in a wizard of oz matter, as the prototypes enabled interactions using the animation feature in Figma to closely mimic how the actual Yolean application behaves [35]. The user performed the same move for all prototypes and was to rank the different prototypes in descending order from 1 to 3 on the approach they preferred. Apart from ranking each prototype and the current solution for each case, the participants for the user test were also encouraged to share thoughts, questions, and reflections through the think-aloud method during the user test to be collected and further analyzed using affinity diagramming.

4.11 Initial User Test Analysis

After completing the initial user test, we analyzed both the quantitative and qualitative data to inform the transition to the high-fidelity prototyping phase. Below, Figure 4.11 visualizes the total count of rankings for each of the three prototypes tested in the distributed error strategy case:

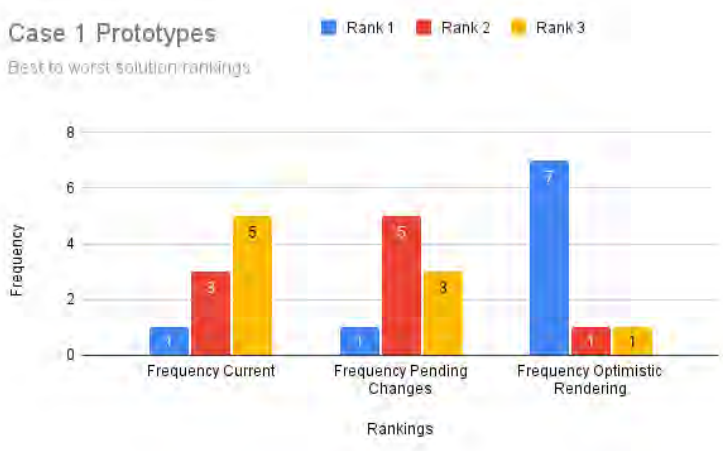

Figure 4.11: Rankings of the case 1 prototypes.

The majority of the test participants rated the optimistic rendering prototype as the best, the pending changes prototype as the second best, and the current behavior as the worst.

Figure 4.12 visualizes the rankings for the different prototypes from the collaborative case:

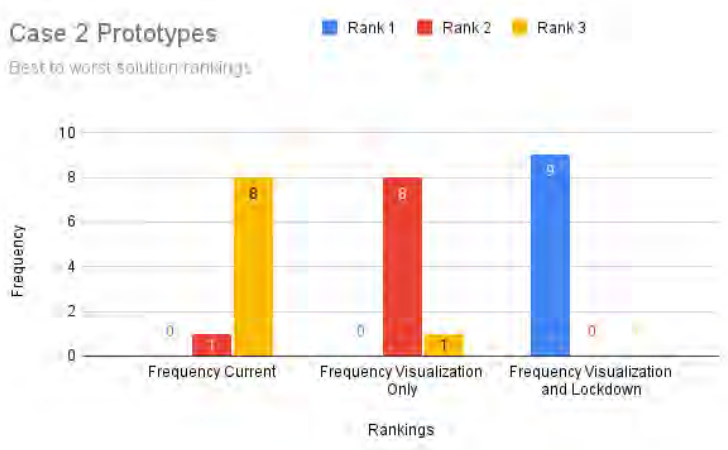

Figure 4.12: Rankings of the case 2 prototypes.

All users considered the lockdown behavior to be the most preferred strategy, while most users considered the visualizations only to be the second best and the current behavior to be the worst strategy.

We integrated the collected qualitative data into our FigJam board to conduct affinity diagramming, aiming to extract further insights to enhance the prototypes in the high-fidelity prototyping phase. Three primary categories emerged: likes, dislikes, and wants. Similar data points were color-coded to denote their significance across the user tests. Figure 4.13 showcases the resulting affinity diagram derived from the qualitative data obtained from the distributed case:

Figure 4.13: Affinity diagram of the data derived from the user test for case 1.

Some key takeaways were that the majority disliked the double rendering behavior of the preview and that the majority liked the optimistic rendering behavior instead. Furthermore, the majority would prefer to have both the optimistic rendering behavior in conjunction with the pending changes list as an opt-in and opt-out mode when needed. Some users also reported that the modal was too small and that there was room for improvements in the visual hierarchy of the information provided for each move action entry in the list.

Figure 4.14 visualizes the resulting affinity diagram from the qualitative data from the collaborative case:

Figure 4.14: Affinity diagram of the data derived from the user test for case 2.

Key takeaways were that the majority of users liked both the visualizations and the lock behavior, disliked the current behavior due to being very frustrating, and would prefer the lock to be activated when dragging a task for a move as well as showing extra information about a user than just a cursor.

4.12 Critique Session

Following the analysis and visual presentation of the findings from our initial user test of the low-fidelity prototypes, we presented the results to some of our stakeholders at Yolean. This group included our advisor, the team lead at Yolean and a senior full-stack developer. Their participation aimed to gain further insights by drawing upon their extensive domain knowledge and understanding of the Yolean application's user base.

In the first case, both stakeholders expressed satisfaction with the integration of optimistic updates featuring modeless feedback for loading states, alongside the introduction of a newly implemented pending changes dialog to offer more comprehensive insights into the application's status. Notably, the test participants preferred an opt-in behavior of the pending changes mode during periods of slower internet or re-connections. Thus, our stakeholder Yolean proposed to execute the final user test

by throttling the internet connection, which would effectively simulate the behavior of a pod replacement and initiate re-connections.

The second case, primarily focused on errors arising during real-time collaboration, prompted discussions regarding the initial visualization concepts and the efficacy of employing a combination of a gag and do nothing method to lock a task when a user is attempting to move an already moving task. Given the event-driven nature of the system, which operates as a distributed system, we discussed how users' clients cannot discern a task-locking status before an event is dispatched to the live server, persisted to Kafka, and then redistributed and propagated to the client. Consequently, the latest user operation most often prevails to be persisted, by the specific logic of the server resolver.

With this constraint brought to light, an alternative strategy was proposed: rather than enforcing component lockdowns, the interface could instead visually denote the individual responsible for the most recent successful write operation to the system's persistence layer. In the context of task movement, this visualization would aim to align users' mental models with the system's operational flows, thus mitigating potential misconceptions regarding unexpected behavior that could be misinterpreted by the user.

In essence, the feedback collected from the critique session underscores the effectiveness of the first case as a means to preemptively communicate the state of write-based user operations, while also providing clear visualization and feedback regarding the success or failure of these operations. Moreover, the adoption of optimistic rendering updates empowers users to seamlessly interact with the interface, in contrast to the previous restrictive 'gag' solution that impeded further actions upon encountering write errors. This transition towards a self-correcting behavior was deemed rational, particularly considering the system's existing provision of undo functionality.

Conversely, the second case serves as a solution geared towards communicating recently executed write operations, thereby facilitating a closer alignment between users' mental models and the underlying architecture of the system. This alignment could potentially enhance users' comprehension of the system's operational dynamics but also streamline the process of identifying and rectifying potential errors that may arise during collaborative efforts within the Yolean application.

4.13 High Fidelity Prototyping

Using the data gathered from the initial user test of the low-fidelity prototypes, and the newly gained insights from the critique session with our stakeholders at Yolean, we decided that for the distributed case, to combine the behavior of optimistic rendering, keeping the loading spinner on each task and combine it with a opt-in mode for view the status for each move in a pending changes list. We also made sure to improve the visual hierarchy, focusing on viability, communication, and efficiency [27]. Furthermore, the pending changes list was moved into the side panel of the Yolean application, to enable the user to opt in and out using the newly added pending changes tab. Furthermore, careful consideration went into designing the

entries to ensure a uniform look and feel with the rest of the interface of the Yolean application.

For the collaborative case, due to technical struggles of implementing a lock behavior, we instead went with another approach for the visualizations, choosing the representation as avatars containing the initials, as the users requested more information than just a cursor in the initial user test. We rendered the avatars in the top bar to communicate active users in a board, as well as indicating who made the latest write to avoid error through appropriate representation, to avoid false understandings for the users, and to improve failure of detecting problems, as a user in the low fidelity user test felt that system behaved buggy or not as it was supposed to [22].

The high-fidelity prototypes for the distributed and collaborative case were implemented on two individual feature branches in the git repository that contains the Yolean application source code. The result of the final prototypes is presented in Figures 5.1 and 5.2.

4.14 Final User Test

After implementing the final prototypes, we advanced to the concluding evaluation phase of the design thinking and requirement process. This phase involved validating our designs through a final user test, aiming to gather insights into our design artifact and draw conclusive findings. As the study was concerned with drawing general conclusions about error-handling strategies for distributed collaborative real-time systems, testing subjects were gathered mainly from other students, as this was the most time-efficient approach within the scope of the thesis.

4.14.1 Demographics of participants

The following pie charts visualize the distribution of age, occupation, field of study, and computer skill for the participants in the final user tests:

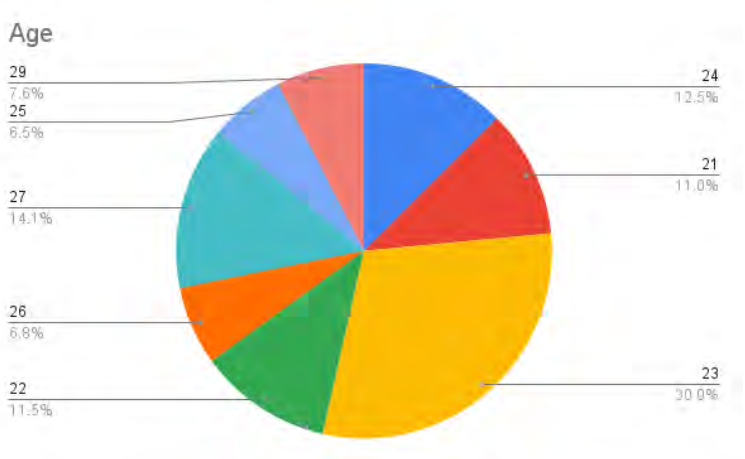

Figure 4.15: Age demographic of participants.

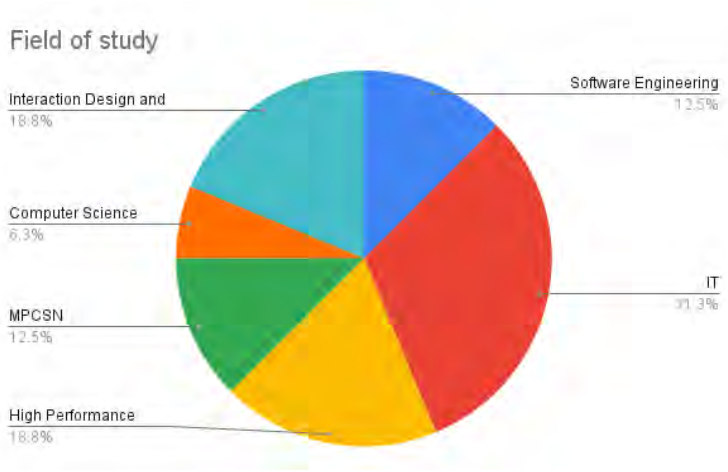

Figure 4.16: Field of study demographic of participants.

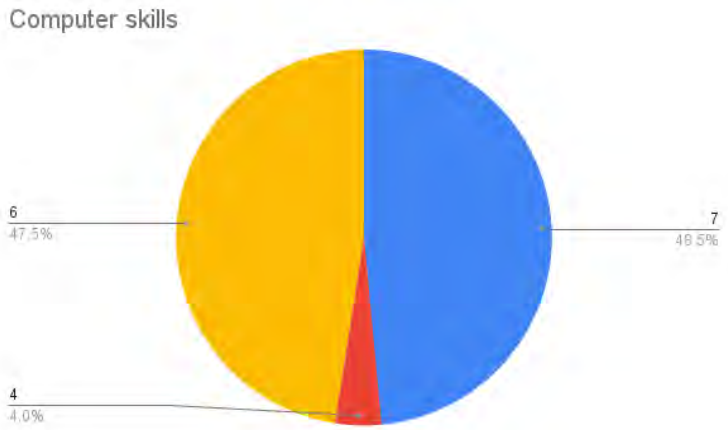

Figure 4.17: Computer skills demographic of participants.

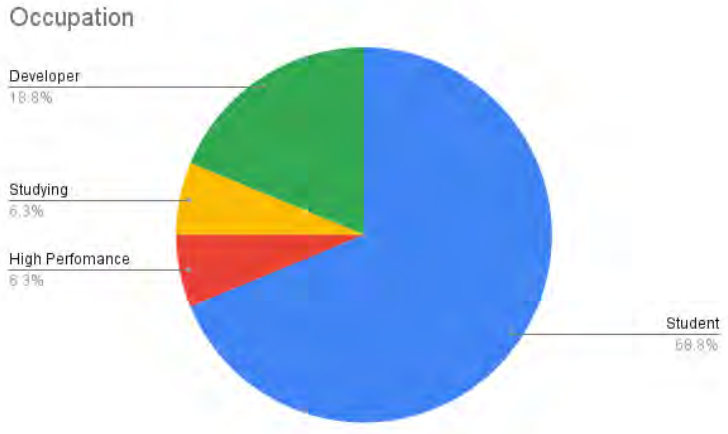

Figure 4.18: Occupation demographic of participants.

4.14.2 Testing procedure

In alignment with the social and ethical considerations outlined in the Theory chapter, we distributed a standardized consent form to ensure compliance with user research guidelines. This consent form was tailored specifically for our study, utilizing the template provided by Chalmers guidelines for handling personal data [36]. Each participant in the user testing session was provided with the consent form, emphasizing the importance of obtaining informed consent before participating in the study.

To address our research questions and gain insights from our design artifact, we designed the final user test to consist of two main parts. The parts consisted of the distributed system case and the collaborative case. The high-fidelity prototypes were used as the independent variable of the user test to measure the change in user response [37]. Drawing inspiration from the A/B design method, we divided the participants of the user test (n=16) into two main groups. The groups consisted of one control group that tested the current state of the Yolean application (n=8) and one test group (n=8) testing the design artifact for the distributed and then the collaborative case. The System Usability Scale was used as the dependent variable to gather data about and measure the difference in the user experience of the interface between the two groups [37], [38]. We began the user test with a brief introduction to the Yolean application and its use cases. Using a survey, we collected the demographic data age, field of study, occupation, and general computer skills. We gave participants written instructions on what task to complete and were encouraged to ask any questions if they felt something was unclear. We collected qualitative data using the Think-aloud protocol, letting the participants express confusion, frustration, and satisfaction during each of the tests they performed. We present the qualitative data in the Results chapter of the thesis. After each task, the participants answered a survey to evaluate their perceived user experience of the Yolean application using the System Usability Scale [37], [38]. After assessing their user experience during the first task, we provided written instructions for task two. After completing the second task, participants again evaluated their experience using the System Usability Scale.

The distributed system test consisted of re-planning a Yolean board in which we instructed participants to change the date of 6 activities in a Yolean board using a throttled network connection. We used the throttled network connection to simulate the same behavior in Yolean as when a pod replacement occurs. The instructions for the distributed case task are listed below:

- Move the start date of task **Lister** from Thursday 02 to Thursday 09
- Move the start date of task **Väggar, Del A** from Friday 03 to Monday 13
- Move the start date of task **Väggar, Del A** from Monday 13 to Friday 10
- Move the start date of task **Kablar, ljus** from Tuesday 23 to Wednesday 01
- Move the start date of task **Måla** from Monday 29 to Tuesday 30
- Move the start date of task **Spakel** from Wednesday 01 to Friday 03

In the development of the collaborative test, we began by brainstorming multiple ideas on how the testing could be effectively conducted within the application. From this, we developed several possible procedures designed to simulate different aspects of real-life usage scenarios. Following this, we discussed these ideas with our stakeholders at Yolean, who provided feedback on the proposed methods. The final test procedure involved the collective re-planning of 14 activities on a Yolean task board. Conducted by two of us overseeing the study, participants were tasked with collaboratively re-planning the board. Each participant received instructions for the move in a different order. This variation was implemented to simulate scenarios where multiple parties share the same objective but approach it differently, a common occurrence in Yolean usage at construction sites, as indicated by insights from our Yolean stakeholders. The instructions for the collaborative case task are listed below:

- Move the start date of task **Elcentraler** from Thursday 02 to Monday 06
- Move the start date of task **Väggar, 5A** from Monday 06 to Wednesday 08
- Move the start date of task **Kontroll** from Friday 03 to Friday 10
- Move the start date of task **Förberedande, fas 2** from Thursday 02 to Wednesday 01
- Move the start date of task **Golvläggning** from Tuesday 30 to Wednesday 01
- Move the start date of task **Ljusfixturer** from Thursday 02 to Friday 03
- Move the start date of task **Avstämning** from Thursday 02 to Friday 03
- Move the start date of task **Riva** from Friday 03 to Monday 06
- Move the start date of task **Reglar** from Thursday 02 to Tuesday 30
- Move the start date of task **Sladdar, Väg del B** from Wednesday 01 to Thursday 09
- Move the start date of task **Trappa, plan 3** from Wednesday 01 to Friday 03
- Move the start date of task **Ombyggnation** from Wednesday 01 to Friday 03
- Move the start date of task **Spakla** from Tuesday 30 to Monday 06
- Move the start date of task **Planering** from Tuesday 30 to Friday 10

4.14.3 SUS Scores and Statistical Significance

Due to the collected SUS data being ordinal, as the framework utilizes rankings of the different questions in the framework, we analyzed the data using the non-parametric test. As we designed the user test to collect two correlated samples for each group, the subjects and the collected SUS data for each group were analyzed using a Mann-Whitney U-test [39]. Due to the smaller sample size of the final user test, we performed hypothesis testing and calculated statistical significance using

the U value and the critical U value for the test. The collected SUS data and the respective results of the Mann-Whitney U-test are presented in the Results chapter.

4.15 Framing Guidelines and Recommendations

After analyzing the data from the final user test, we summarized our findings into an initial framework designed to facilitate further research on similar systems in the future. This framework aims to provide a systematic approach to exploring error-handling strategies for collaborative distributed real-time systems, allowing for refinement and expansion over time, thereby contributing new knowledge to the field. The framework consists of five main guidelines, which we present in the Results chapter of this thesis. It was created by synthesizing the knowledge gained from the literature study, the developed error awareness matrix, the design process, and the consecutive user tests.

5

Results

This chapter presents our results as the final iteration of our design artifact, consisting of the final design solution of the high-fidelity prototype and the presentation and statistical analyses of the quantitative and observations from the qualitative data collected from the final user test.

5.1 High Fidelity Prototype

Figure 5.1 visualizes the final prototype for the distributed case implemented in the Yolean application:

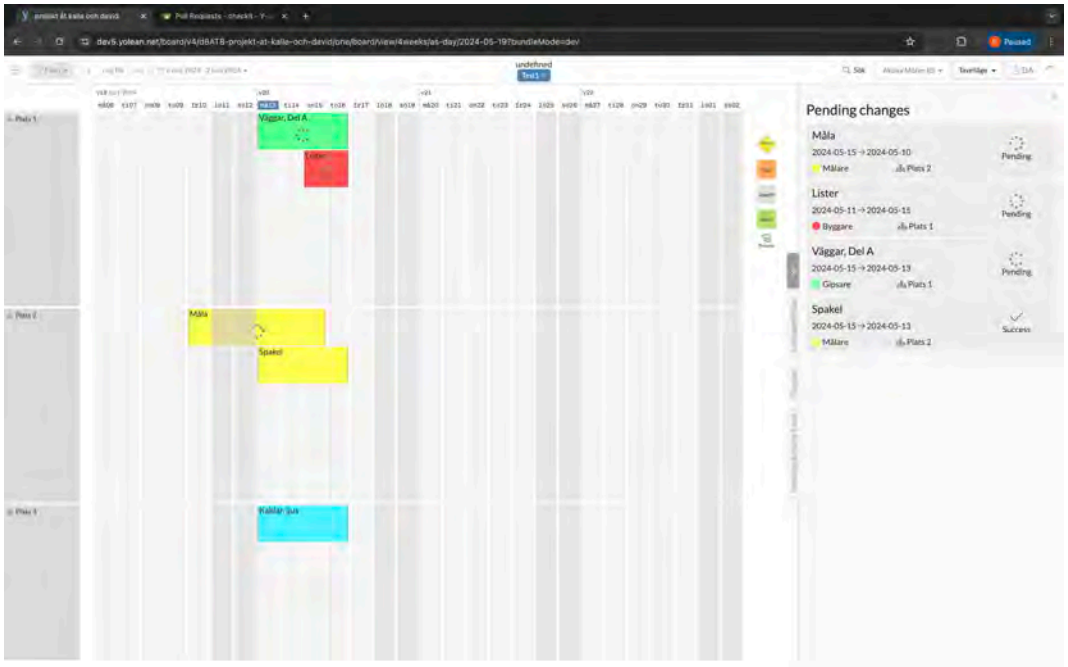

Figure 5.1: Screenshot of the final prototype for the distributed case.

The pending changes list has been moved into the sidebar to enable opt-in and out behavior, with improved visual hierarchy and adherence to the general look and feel of the Yolean application. The figure depicts multiple pending changes, with their unsynced state indicated by the loading spinner and the text element that indicates

the status. Furthermore, the figure visualizes an entry for a successful state update, with the success icon and respective text. Furthermore, the figure visualizes the optimistically rendered tasks **Måla**, **lister**, and **Väggar, Del A**. The high-fidelity prototype for the distributed system case differs from the current state of Yolean in the introduced pending changes log, as well as the optimistic rendering behavior of pending tasks, pivoting the error handling strategy from utilizing forcing functions and a gagging mechanism to a combination of the warn, let's talk about it, do nothing and self-correct strategies [22].

Figure 5.2 visualizes the final prototype of the collaborative case implemented in the Yolean application:

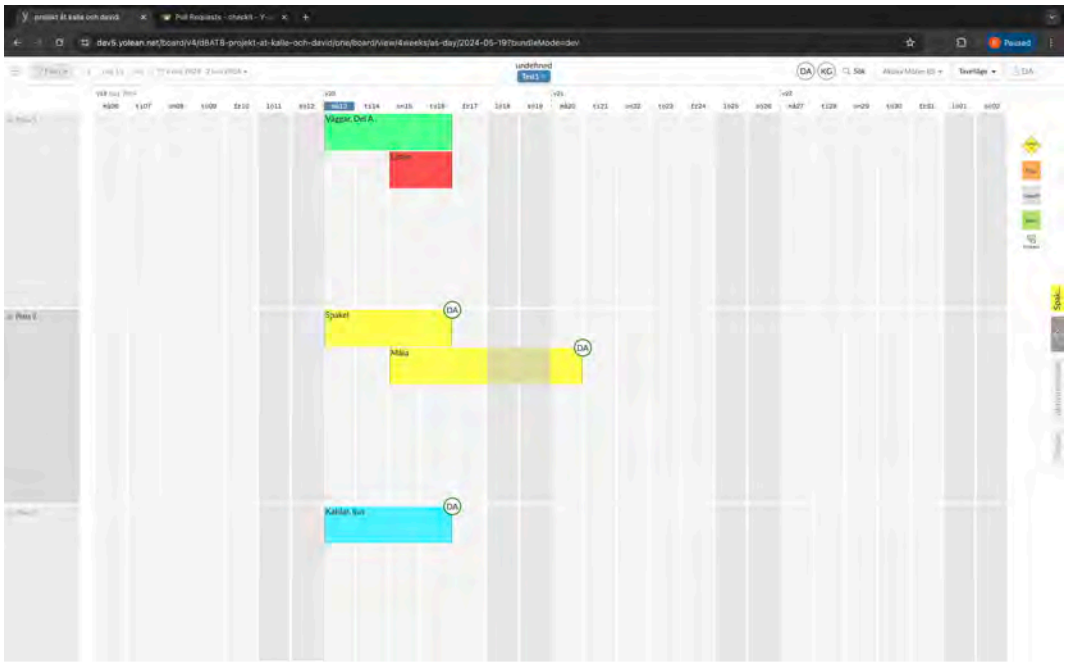

Figure 5.2: Screenshot of the final prototype for the collaborative case.

The element containing the avatars that communicate the active users of a Yolean board is rendered in the top bar. The user's avatar indicating the latest successful move renders at the upper right corner of the affected task. The high-fidelity prototype for the collaborative real-time case differs from the current state of Yolean with the introduction of the present users and the rendering of avatars on the latest move operations, intending to create a more appropriate representation, avoiding false understandings and mitigating failure in detecting problems [22].

5.2 Final User Test

The following two sections present the results from the statistical analysis of the quantitative data gathered through the System Usability Scale. The later section presents the qualitative data gathered using the think-aloud protocol through observations from the final user test.

5.2.1 SUS Scores and Statistical Significance

The result of the participants' ratings using the System Usability Scale for both the control and test group of the final user test is presented in the tables below. Furthermore, we present the results of the Mann-Whitney U test conducted on the data gathered through the SUS scale for both the distributed and the collaborative error-handling strategy.

Participant	**SUS score**
Participant 1	35
Participant 2	35
Participant 3	60
Participant 4	72.5
Participant 5	32.5
Participant 6	37.5
Participant 7	42.5
Participant 8	30

Table 5.1: The SUS scores for the control group in the distributed case experiment.

Participant	**SUS score**
Participant 9	80
Participant 10	77.5
Participant 11	70
Participant 12	82.5
Participant 13	82.5
Participant 14	82.5
Participant 15	75
Participant 16	85

Table 5.2: The SUS scores for the test group in the distributed case experiment.

The results of the Mann-Whitney U test test indicated that there is significant difference between Control group, SUS score (Mdn = 36.25 ,n = 8) and Test Group, SUS score (Mdn = 81.25 ,n = 8), p = .01, U = 1. The critical value of U at $p < .01$ is 9. Therefore, the result is significant at $p < .01$.

Participant	SUS score
Participant 1	30
Participant 2	27.5
Participant 3	82.5
Participant 4	82.5
Participant 5	30
Participant 6	45
Participant 7	80
Participant 8	35

Table 5.3: The SUS scores for the control group in the collaborative case experiment.

Participant	SUS score
Participant 9	85
Participant 10	85
Participant 11	77.5
Participant 12	77.5
Participant 13	72.5
Participant 14	85
Participant 15	70
Participant 16	82.5

Table 5.4: The SUS scores for the test group in the collaborative case experiment.

The results of the Mann-Whitney U test test indicated that there is significant difference between Control group, SUS score (Mdn = 40 ,n = 8) and Test Group, SUS score (Mdn = 80 ,n = 8), p = .05, U=13. The critical value of U at $p < .05$ is 15. Therefore, the result is significant at $p < .05$.

5.2.2 Observations

In the final user test, observations were collected through the think-aloud protocol to further evaluate the effectiveness of the design artifact, focusing on both the distributed and collaborative error-handling strategies. These observations provide valuable insights into the user experience during the final user test, highlighting the differences in user perceptions and behaviors between the control and test groups for both the distributed and collaborative cases. The following sections detail the specific observations and insights gathered from each group in both cases:

Distributed Case - Control Group

- Confusion and Uncertainty: Participants struggled to understand the application's state and the persistence of their actions, leading to low confidence in their interactions.
- Inconsistency in Rendering: Previews sometimes appeared and disappeared unpredictably, causing confusion. Tasks occasionally failed to move despite

drag actions.

- Frustration with Correction: Inability to correct mistakes easily due to system limitations led to frustration. Participants felt the system was cluttered and buggy.

Distributed Case - Test Group

- Immediate Task Movement: Participants appreciated the ability to move tasks immediately, especially when placed incorrectly.
- Usefulness of Pending Changes Tab: The pending changes tab provided clarity on the application's state, increasing participants' confidence in their actions. Some noted its potential for error tracking and additional verification.

Collaborative Case - Control Group

- Lack of Visualization: Participants found it challenging to follow changes without any visual cues, especially regarding the actions of others.
- Confusion with Re-rendering: Re-rendering without indicators led to confusion and a perception of the system as buggy and unintuitive.

Collaborative Case - Test Group

- Appreciation for Avatars: Participants welcomed the introduction of avatars, which helped communicate others' changes. They understood why tasks were moved and re-rendered.
- Placement Confusion: Some participants noted occasional confusion when others' changes overlapped with their own tasks, affecting their understanding of the application's state.

5.3 Recommendations and Guidelines

We present a framework that provides a structured approach to identifying, handling, and mitigating errors in collaborative distributed real-time systems based on the findings from our research. The framework consists of five main guidelines with additional recommendations: Identifying and Classifying Error States, Designing for Error Prevention and Handling, Implementing Error Handling Strategies, Enhancing User Communication and Iterative Design, and User Testing. These guidelines act as a basis for structuring the process of exploring error-handling strategies to improve the end-user experience through enhanced usability and reliability.

5.3.1 Identifying and Classifying Error States

Identifying why and how a system enters an erroneous state is important for understanding the problem space and identifying possible solution strategies.

- **Determine Reason Behind Error**

  - **Description:** Determine whether the erroneous state is caused by the system design or a mistake or slip by the user.
  - **Implementation steps:**
    * Use the subset of the CSNI Taxonomy to make an initial classification of the erroneous state.

    Example: In Yolean, an erroneous state may occur during a pod replacement, a direct consequence of the distributed architecture. Furthermore, a human-induced error state could occur due to slips or mistakes during collaborative re-planning.

- **Error awareness matrix**
  - **Description:** Utilize an error awareness matrix to categorize potential error states based on the awareness levels of both the client and server components.
  - **Implementation steps:**
    * Identify error states and classify them into correct awareness states from the awareness matrix seen in Figure 1.1.
    * Use this classification to explore appropriate error-handling strategies.

    Example: In Yolean, errors related to unsynced states during slow network conditions or pod replacement can be categorized to develop appropriate strategies and representations in the interface.

5.3.2 Designing for Error Prevention and Handling

Designing for error prevention and handling is paramount to ensure a smooth user experience and avoid mistakes and slips. By utilizing clear representations and providing comprehensive information, designers can align the system with users' mental models. This alignment reduces errors, enhances usability, and builds user confidence, leading to a more reliable and efficient system.

- **Appropriate Representation**
  - **Description:** Ensure system representations are unambiguous to minimize errors. Appropriate representations mitigate the problem of level and failure to detect problems due to aligning the user's mental model with the implementation of the system.
  - **Implementation steps:**
    * Use visual representations (e.g., icons, direct manipulation interfaces) instead of text-based ones to reduce the likelihood of user errors.
    * Provide contextual information to help users understand the implications of their actions

Example: In the high-fidelity prototype for the collaborative case, users' write operations are represented with Avatars, aligning the users' mental model of the system with the event-driven nature of the architecture.

- **Avoiding False Understandings**

  Description: Prevent misconceptions about system operations by providing comprehensive information to users, to mitigate the problem of level and failure to detect problems due to aligning the mental model of the user with the system.

  - **Implementation Steps**:
    - Offer clear explanations and feedback for user actions.
    - Design user interfaces that highlight critical information and potential error points.

  Example: In the high fidelity prototype for the distributed case, the pending changes log allows the user to verify that their change has persisted.

5.3.3 Implementing Error Handling Strategies

Implementing effective error-handling strategies is essential for creating robust and user-friendly systems. These strategies enhance user experience by automatically resolving errors and keeping users informed. The strategies reduce user frustration, improve system reliability, and ensure smoother interactions for a better-perceived user experience.

- **Caution with forcing functions and gag mechanisms**

  Description: Be cautious of implementing mechanisms that prevent users from proceeding when an error is detected.

  - **Implementation Steps**:
    - Oversee and identify the system's current forcing functions and gag mechanisms.
    - Consider how this mechanism could be less restrictive for the user or consider alternative error-handling strategies.

  Example: In the high-fidelity prototype for the distributed case, optimistic rendering allows users to correct mistakes and slips, even when encountering re-connections.

- **Self-correcting systems**

  Description: Design systems that automatically resolve errors without user intervention.

  - **Implementation Steps**:

    * Implement system-level correcting features and provide robust undo functionalities.
    * Use logs to trace and resolve errors autonomously.

  Example: The event-driven nature of Yolean continuously tries to resolve the state of the application. The newly introduced pending changes log enables the user to trace what changes made it through to persistence in conjunction with existing undo functionality.

- **Optimistic Updates**

  Description: Allow for optimistic updates where changes are temporarily accepted while awaiting confirmation from the server.

  – **Implementation Steps**:
    * Reflect changes in the user interface immediately, but mark them as pending.
    * Use visual indicators to inform users of the pending state.

  Example: In the high-fidelity prototype for the distributed case in Yolean, optimistic updates with clear pending change indicators help manage user expectations during an unsynced state.

5.3.4 Enhancing User Communication

Effective system design hinges on clear communication of system states and errors to users to help users quickly and easily identify errors.

- **Visual Indicators**

  Description: Use visual cues to communicate system states and errors.

  – **Implementation Steps**:
    * Employ high-contrast formatting, bold text, and red styling for error indicators.
    * Ensure accessibility by using multiple indicators beyond visual cues.

  Example: In the high-fidelity prototype for the distributed case, the pending changes log is designed with a clear visual hierarchy and with appropriate icons and colors for user actions that failed to persist. The loading spinner combined with allowing optimistic rendering communicates an unsynced state and warns of possible future errors.

5.3.5 Iterative Design and User Testing

Iterative design and user testing are essential components of developing error-handling strategies that align with users' preferences.

- **Description**: Employ an iterative design process with frequent user testing to refine error-handling strategies.
  - **Implementation Steps**:
    * Conduct multiple rounds of prototyping and user testing.
    * Collect both qualitative and quantitative data to understand user preferences.

  Example: Combining research through design, design thinking, and requirements engineering allowed for designing solutions grounded in theory and consecutive user feedback.

- **Think-Aloud Protocols**

  Description: Use think-aloud protocols during testing to gain deeper insights into user thought processes.
  - **Implementation Steps**:
    * Encourage users to verbalize their thoughts while interacting with the prototype.
    * Analyze the feedback to identify pain points and areas for improvement.

  Example: Throughout the user tests, the think-aloud protocol helped uncover contradictory data. For instance, the users from the user tests disliked forcing functions and gag mechanisms for the investigated distributed case but wished for it for the collaborative one.

By following this framework, developers and researchers can systematically address error handling in collaborative distributed real-time systems, ensuring a more robust and user-friendly experience. This framework provides a comprehensive approach that can be generalized and applied to similar systems, contributing valuable knowledge to the field.

6

Discussion

The following chapter aims to answer the research questions introduced in the introduction chapter and to provide reflection about the process and outcomes as a whole, opening up for possible further aspects for future work.

6.1 Error States

The developed error awareness matrix, as presented in Figure 1.1, serves as a foundational tool for understanding the intricate landscape of potential error states within collaborative distributed real-time systems, such as the Yolean platform. This matrix offers a high-level overview that aids in reasoning about the diverse range of errors that these systems can encounter, adding further context and insight into the problem space for errors classified through utilizing the subset of the CSNI Taxonomy [24]. Given the focus of this thesis on interaction design, particular emphasis was placed on error scenarios where either the server or client, or both, find themselves in a state where errors are either unknown or unacknowledged. This attention stems from the recognition that errors falling within the first case of the matrix, those where both server and client are aware of the error, can often be handled through robust coding practices. However, complexities arise when errors occur in scenarios where one or both parties are unaware of the error's existence, warranting exploration of handling and presentation strategies to improve the user experience and reliability of the system when working in such error states.

6.2 Presentation and Error Handling Strategies

Using the subset of the CSNI taxonomy classification framework and the developed error awareness matrix as a foundation, we selected and classified technical implications to begin exploring potential presentation strategies to improve the user experience of collaborative distributed real-time systems like Yolean [24]. Our focus was on investigating the distributed systems' technical aspects of Yolean, particularly the operation of multiple server pods, which often are replaced during scaling or when encountering issues [12]. Additionally, we examined a collaborative case where errors arose due to human factors, affecting the server and client without their awareness. Through theory, we found various approaches for handling and presenting errors that could be applied to systems such as Yolean, such as forcing functions, gagging,

warning, doing nothing, self-correction, initiating discussions, and providing guidance [22]. We opted to explore contrasting strategies to those currently used in the Yolean application, allowing for a comparative analysis of their effects on addressing issues such as the problem of level, failure to detect problems, and avoiding false understandings inherent in such systems. While we did not explore all theoretical handling and presentation strategies, low-fidelity prototyping enabled us to delve into a subset of them, aligning the ideas for the prototypes with user feedback and preferences.

6.3 Broader Insights in Error Handling

The research as a design process in conjunction with design thinking and requirements engineering provided an effective method for sparking insights, discussion, and reflection in the broader area of error handling. Through the collected results of the thesis, the design artifact initiated discussions on the preferred theoretical methods of handling and presenting errors by enabling comparisons with contrasting strategies employed by the current Yolean application. Overall, users tended to prefer approaches such as 'let's talk about it,' 'do nothing,' 'warn,' and 'self-correct,' which were reflected in the implementations of the final design solutions [22]. Interestingly, users also expressed a preference for a 'lockdown' mechanism in the collaborative case, despite reporting frustration with similar behaviors in the distributed case. Unfortunately, technical difficulties prevented further iteration and implementation of the lockdown functionality in the final high-fidelity prototype. Exploring the reasons behind this discrepancy in user preference based on the error case from the awareness matrix would be an intriguing avenue for future investigation. Furthermore, the qualitative data from the user tests signify that the pending changes and avatars made a significant difference in aligning their mental models of the interface with a reasonable abstraction of the system's flow and functions [39]. This, in conjunction with the difference in SUS scores and reported remarks on how users had more confidence in the state of the application, suggests that the extra information added through the 'let's talk about it' approach, in conjunction with an accurate representation of the system, seemed to diminish the problem of level, failure to detect problems, and avoiding false understanding [22], [37], [38].

6.4 Guidelines and Recommendations for Error Handling

The framework presented in the recommendations and guidelines section of the result provides a structured framework for addressing error handling in collaborative distributed real-time systems. This framework offers practical, actionable steps for identifying, handling, and mitigating errors, grounded in theoretical insights and empirical findings from our research [22]–[24], [27]. It includes guidelines for determining the reason behind errors, using an error awareness matrix, designing for error prevention and handling, implementing error handling strategies, and enhanc-

ing user communication. These guidelines serve as a general knowledge contribution that can be useful for others facing similar problems. By outlining actionable steps and examples, we aim to provide valuable tools for developers and researchers working on similar projects.

6.5 Reflections on Methodology, Reliability and Validity

The results of the final user test indicate a significant statistical increase in the perceived usability of our prototype in both the distributed system and the collaborative case. The significant statistical increase provides confidence that our solutions represent an improvement over the current solutions implemented in Yolean [37]–[39]. However, it is important to acknowledge the relatively low sample size in the study (n=16) when drawing broader conclusions, as lower sample sizes carry a higher risk of bias. Additionally, the demographic of our test participants must be taken into consideration and its potential impact on the study. The fact that we primarily recruited participants at campus and the Yolean office, due to their accessibility, resulted in most participants sharing similar backgrounds and ages. It is crucial to consider the narrow demographic of our participants when applying our findings to the broader population. While our research primarily focused on exploring general knowledge within the field, one key aspect differs from the typical users of the Yolean. The participants of the user test were first-time users of the application. Thus, the inexperience in using the application might have led to greater appreciation for the new solutions provided through the design artifact, and that long-time user would not necessarily reach the same level of appreciation due to already being familiar with the application and its behavior. However, at this stage, these reflections stem from conjecture, and we would need to verify this with user tests with real users from the construction sector.

Due to a constrained implementation timeline, the project only assessed a subset of error-handling strategies. We primarily selected these strategies based on the insight and knowledge from early interviews and initial user tests. However, this limited the study's results and restricted the breadth of its findings. With more feasible iterations, the study could have evaluated additional error-handling strategies, potentially gaining deeper insights into identifying the optimal set of strategies. While the study's findings provide valuable insights within the defined scope, there is potential for further exploration. If we were to extend the scope or redo the project entirely, we would like to compare and evaluate the full range of error mitigation and error handling strategies introduced in the theory chapter. Covering the whole set of strategies would be beneficial in gaining deeper insights to craft more refined guidelines. Furthermore, the process would have been expanded with further iterations to test the full range of high-fidelity prototypes, allowing for comparison and further analysis throughout the project. Lastly, we would have liked to bring down the scope to the collaborative and distributed case at an earlier stage of the project, which was difficult due to the vast landscape of the area where we are conducting research. Bringing down the scope at an earlier phase would possibly have enabled

further iterations during the prototyping phases, which were the vast knowledge and insights verified during user testing.

In the final user test, the combined use of quantitative data from the System Usability Scale (SUS) and qualitative data from the think-aloud protocol allowed us to achieve a comprehensive evaluation of our design artifacts. The quantitative data from the SUS provided an objective assessment, offering measurable data on the usability and efficiency of the designs. This objective measurement was useful for systematically identifying specific strengths and weaknesses. On the other hand, the qualitative data from the think-aloud protocol enabled us to delve deeper into the user experience by capturing participants' thoughts and feelings as they interacted with the design artifacts. This method revealed nuances and contextual factors that numerical data could not, giving us a better understanding of user satisfaction and the emotional impact of our designs. Together, these methods provided a balanced approach, ensuring the quantitative aspects of usability while also considering the qualitative experiences of the users.

6.6 Future work

While the final prototypes incorporate enough functionality to explore the user experience of different approaches in error handling and presentation strategies, there is room for further exploration of other potential user actions, as moving tasks are only a small subset of possible actions a user can take in the Yolean application. Furthermore, for further exploration of the distributed case, it would be interesting to see how the pending changes log would fend in adding incoming changes as a variable. For the collaborative case, it would be interesting to see how it fairs when users have different views than each other, using the extensive filter capabilities of Yolean, where tasks might not be in view that another user interacts with. Another aspect to consider in the collaborative case is investigating how turn-taking could influence the perceived user experience. Although turn-taking is not a typical usage pattern for the Yolean application, it presents a contrasting area to study within the broader context of error handling in collaborative systems. Moreover, there is potential for further refinement of the framework introduced in the recommendations and guidelines section of the Result. A set of more comprehensive rules or guidelines for classifying error states could enhance the usability and applicability of the framework, allowing for more widespread adoption and generalization of its principles, as the framework currently exists in its infancy. Furthermore, future research endeavors could focus on comparing a comprehensive array of error-handling approaches rather than solely contrasting strategies. By examining a broader spectrum of design methodologies, it would be possible to draw more generalized conclusions about the approaches preferred by end users and their effectiveness in improving user experience.

7

Conclusion

In conclusion, this study has contributed valuable insights into the landscape of error handling and presentation strategies within collaborative distributed real-time systems, exemplified by the Yolean platform. Through the literature study, development, and application of an error awareness matrix, alongside the exploration of handling and presentation strategies and user testing, we have advanced understanding within this area of system design summarized in an initial framework for exploring error handling strategies.

The findings underscore the importance of considering technical and human factors in error handling, particularly in scenarios where errors may go unnoticed by one or more parties involved. By delving into contrasting strategies and iteratively refining design solutions based on user feedback, we have identified promising approaches for improving the user experience and reliability of such systems.

Reflections on methodology highlight the need for iterative refinement and broader sampling in future research. While the study's findings offer significant insights within its defined scope, there remains the opportunity for further exploration. Future work could delve into additional user actions, refine the framework, and compare a more comprehensive array of error-handling approaches to draw generalized conclusions.

Overall, this study sets the stage for continued advancements in the design and implementation of error-handling strategies in collaborative distributed real-time systems, paving the way for enhanced usability, reliability, and user satisfaction in future iterations of such platforms, emphasizing the effectiveness of research as design in conjunction with design thinking and requirements engineering to effectively explore new solutions within the field.

Bibliography

[1] *Yolean*, https://www.yolean.com/, Accessed: January 31, 2024.

[2] M. Van Steen and A. S. Tanenbaum, "A brief introduction to distributed systems," *Computing*, vol. 98, pp. 967–1009, 2016.

[3] *Docker*, https://www.docker.com/, Accessed: January 31, 2024.

[4] *Kubernetes*, https://kubernetes.io/, Accessed: January 31, 2024.

[5] *Amazon Web Services (AWS) Microservices*, https://aws.amazon.com/microservices/, Accessed: January 31, 2024.

[6] *Apache Kafka*, https://kafka.apache.org/, Accessed: January 31, 2024.

[7] Mozilla Developer Network, *WebSockets API*, https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API, Accessed: January 31, 2024.

[8] E. Hollnagel, "The phenotype of erroneous actions: Implications for hci design," *Human-computer interaction and complex systems*, pp. 73–121, 1991.

[9] S. J. Park, C. M. MacDonald, and M. Khoo, "Do you care if a computer says sorry? user experience design through affective messages," in *Proceedings of the designing interactive systems conference*, 2012, pp. 731–740.

[10] Y. Inal and N. Ozen-Cinar, "Achieving a user friendly error message design: Understanding the mindset and preferences of turkish software developers," in *Design, User Experience, and Usability: Novel User Experiences: 5th International Conference, DUXU 2016, Held as Part of HCI International 2016, Toronto, Canada, July 17–22, 2016, Proceedings, Part II 5*, Springer, 2016, pp. 377–385.

[11] R. Krause, *How to report errors in forms: 10 design guidelines*, Nielsen Norman Group, Feb. 2019. [Online]. Available: https://www.nngroup.com/articles/errors-forms-design-guidelines/.

[12] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Kubernetes as an availability manager for microservice applications," *arXiv preprint arXiv:1901.04946*, 2019.

[13] M. Randles, D. Lamb, E. Odat, and A. Taleb-Bendiab, "Distributed redundancy and robustness in complex systems," *Journal of Computer and System Sciences*, vol. 77, no. 2, pp. 293–304, 2011.

[14] M. Colnari and W. A. Halang, "Exception handling and predictability in hard real-time systems," in *SAFECOMP93: The 12th International Conference on Computer Safety, Reliability and Security 12*, Springer, 1993, pp. 371–378.

[15] U. M. Borghoff, J. H. Schlichter, U. M. Borghoff, and J. H. Schlichter, "Fundamental principles of distributed systems," *Computer-Supported Cooperative Work: Introduction to Distributed Applications*, pp. 3–85, 2000.

[16] R. Stroud, "Transparency and reflection in distributed systems," in *Proceedings of the 5th workshop on ACM SIGOPS European workshop: Models and paradigms for distributed systems structuring*, 1992, pp. 1–5.

[17] E. A. Brewer, "Towards robust distributed systems," in *PODC*, Portland, OR, vol. 7, 2000, pp. 343 477–343 502.

[18] S. Gilbert and N. Lynch, "Perspectives on the cap theorem," *Computer*, vol. 45, no. 2, pp. 30–36, 2012.

[19] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi, "The evolution of distributed systems towards microservices architecture," in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, IEEE, 2016, pp. 318–325.

[20] O. Schmid, A. Lisowska Masson, and B. Hirsbrunner, "Real-time collaboration through web applications: An introduction to the toolkit for web-based interactive collaborative environments (twice)," *Personal and Ubiquitous Computing*, vol. 18, pp. 1201–1211, 2014.

[21] Google LLC, *Firebase*, `https://firebase.google.com/`, Accessed 2024.

[22] C. Lewis and D. A. Norman, "Designing for error," in *Readings in human–computer interaction*, Elsevier, 1995, pp. 686–697.

[23] J. A. Bargas-Avila, G. Oberholzer, P. Schmutz, M. de Vito, and K. Opwis, "Usable error message presentation in the world wide web: Do not show errors right away," *Interacting with Computers*, vol. 19, no. 3, pp. 330–341, 2007.

[24] J. Rasmussen, O. Pedersen, G. Mancini, A. Carnino, M. Griffon, and P. Gagnolet, "Classification system for reporting events involving human malfunctions," *Report EUR*, 1981.

[25] A. Cooper, R. Reimann, D. Cronin, and C. Noessel, *About face: the essentials of interaction design*, 4th ed. John Wiley & Sons, 2014.

[26] R. Molich and J. Nielsen, "Improving a human-computer dialogue," *Communications of the ACM*, vol. 33, no. 3, pp. 338–348, 1990.

[27] T. Neusesser and E. Sunwall, *Error message guidelines*, Nielsen Norman Group, May 2023. [Online]. Available: `https://www.nngroup.com/articles/error-message-guidelines/` (visited on 02/15/2024).

[28] G. Beretta, H. Burkhart, P. Fink, *et al.*, "Xs-1: An integrated interactive system and its kernel," in *Proceedings of the 6th international conference on Software engineering*, 1982, pp. 340–349.

[29] R. Maria, *Ethical maturity in user research*, Nielsen Norman Group. [Online]. Available: `https://www.nngroup.com/articles/user-research-ethics/` (visited on 03/09/2023).

[30] T. Robertson, "Ethical issues in interaction design," *Ethics and Information Technology*, vol. 8, pp. 49–59, Jun. 2006. DOI: `10.1007/s10676-005-8308-3`.

[31] J. Zimmerman, J. Forlizzi, and S. Evenson, "Research through design as a method for interaction design research in hci," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2007, pp. 493–502.

[32] A. Sutcliffe, *Requirements Engineering*. IxDF. [Online]. Available: `https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/requirements-engineering` (visited on 02/02/2024).

[33] R. Razzouk and V. Shute, "What is design thinking and why is it important?" *Review of educational research*, vol. 82, no. 3, pp. 330–348, 2012.

[34] H.-P. Institute, *The six phases of the design thinking process*, hpi.de. [Online]. Available: `https://hpi.de/en/school-of-design-thinking/design-thinking/background/design-thinking-process.html` (visited on 02/07/2024).

[35] B. Hanington and B. Martin, *Universal methods of design expanded and revised: 125 Ways to research complex problems, develop innovative ideas, and design effective solutions*. Rockport publishers, 2019.

[36] *Handling of personal data*, Chalmers University of Technology. [Online]. Available: `https://www.chalmers.se/en/education/your-studies/masters-and-bachelors-thesis/handling-of-personal-data/` (visited on 03/09/2023).

[37] I. S. MacKenzie, "Human-computer interaction: An empirical research perspective," 2012.

[38] J. R. Lewis and J. Sauro, "Item benchmarks for the system usability scale.," *Journal of Usability Studies*, vol. 13, no. 3, 2018.

[39] P. Billiet, *The mann-whitney u-test –analysis of 2-between-group data with a quantitative response variable*, 2003. [Online]. Available: `https://psych.unl.edu/psycrs/handcomp/hcmann.PDF`.

A

Initial Affinity Diagramming and Theory Mapping

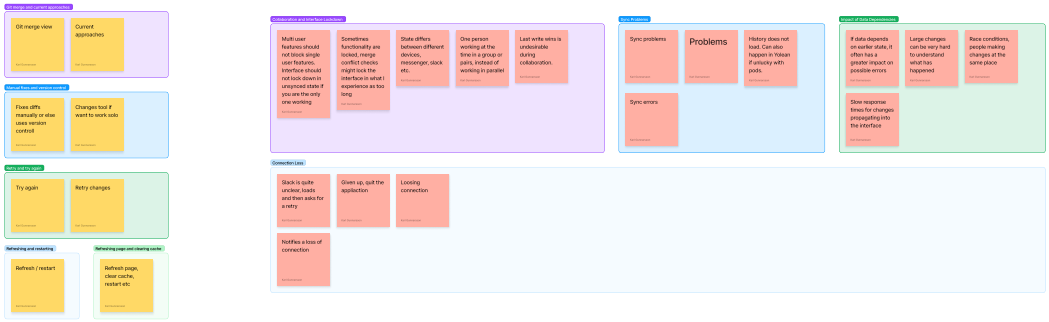

Figure A.1: Part 1 of the initial affinity diagram.

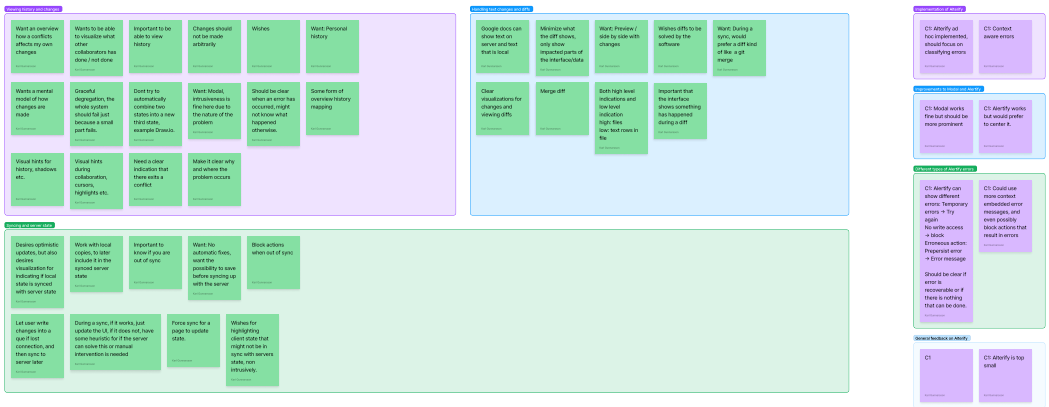

Figure A.2: Part 2 of the initial affinity diagram.

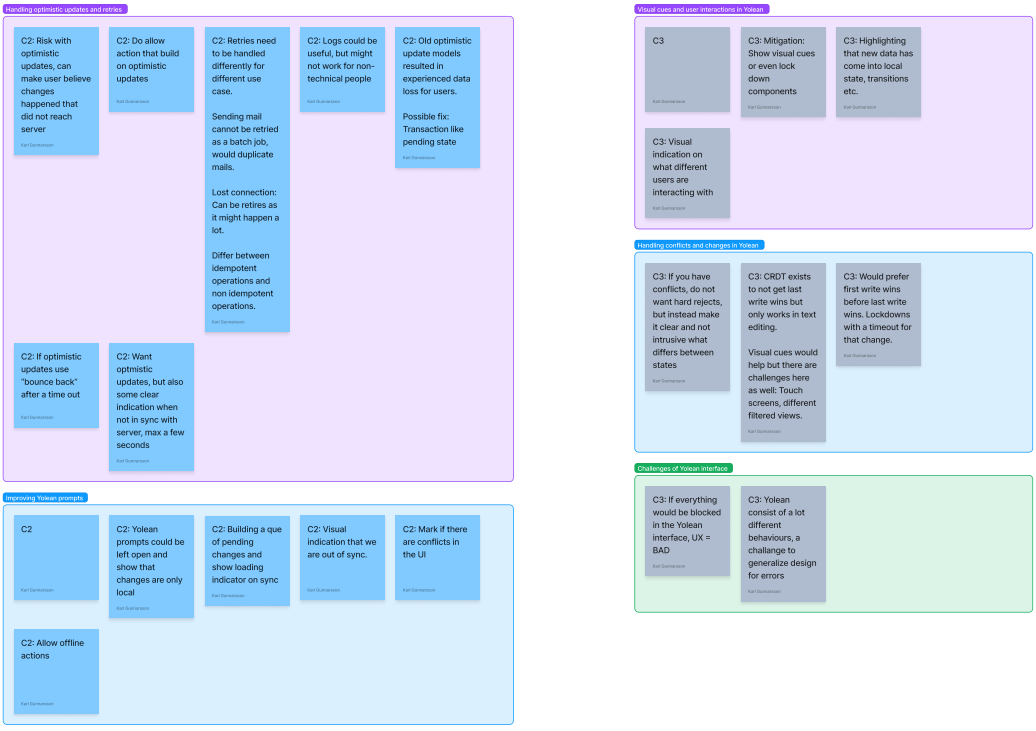

Figure A.3: Part 3 of the initial affinity diagram.

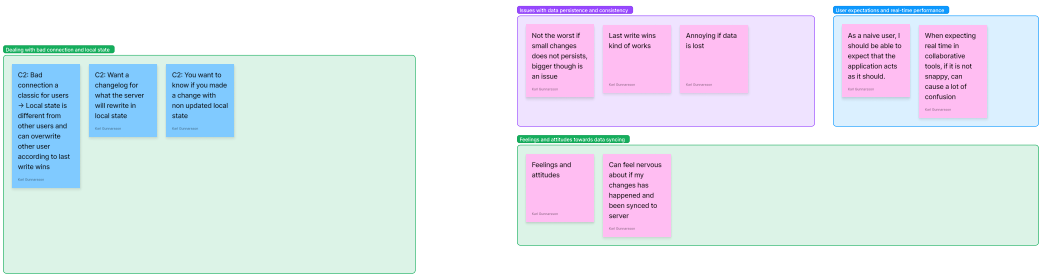

Figure A.4: Part 4 of the initial affinity diagram.

Figure A.5: Interview answers mapped onto the Forcing Function and Warn strategy.

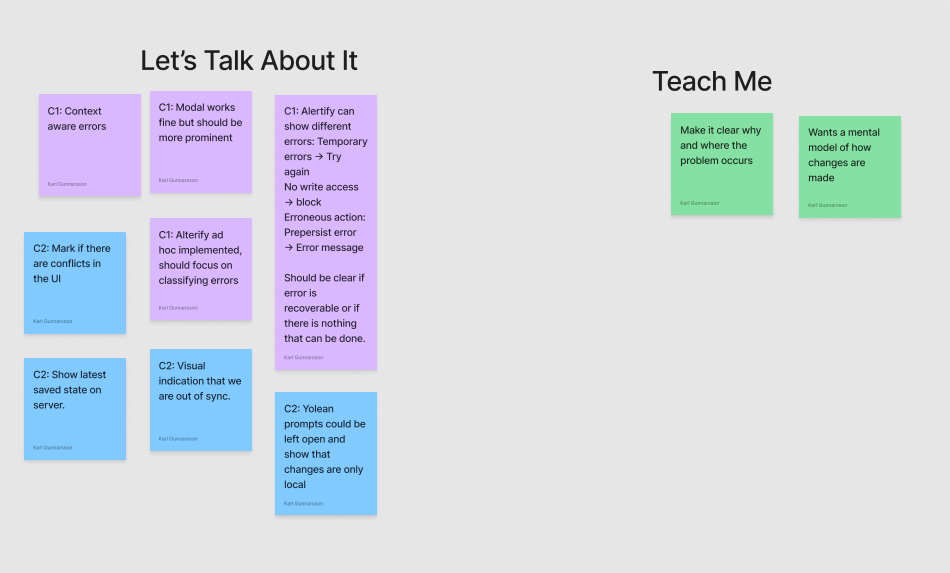

Figure A.6: Interview answers mapped onto the Let's Talk About it and Teach Me strategy.

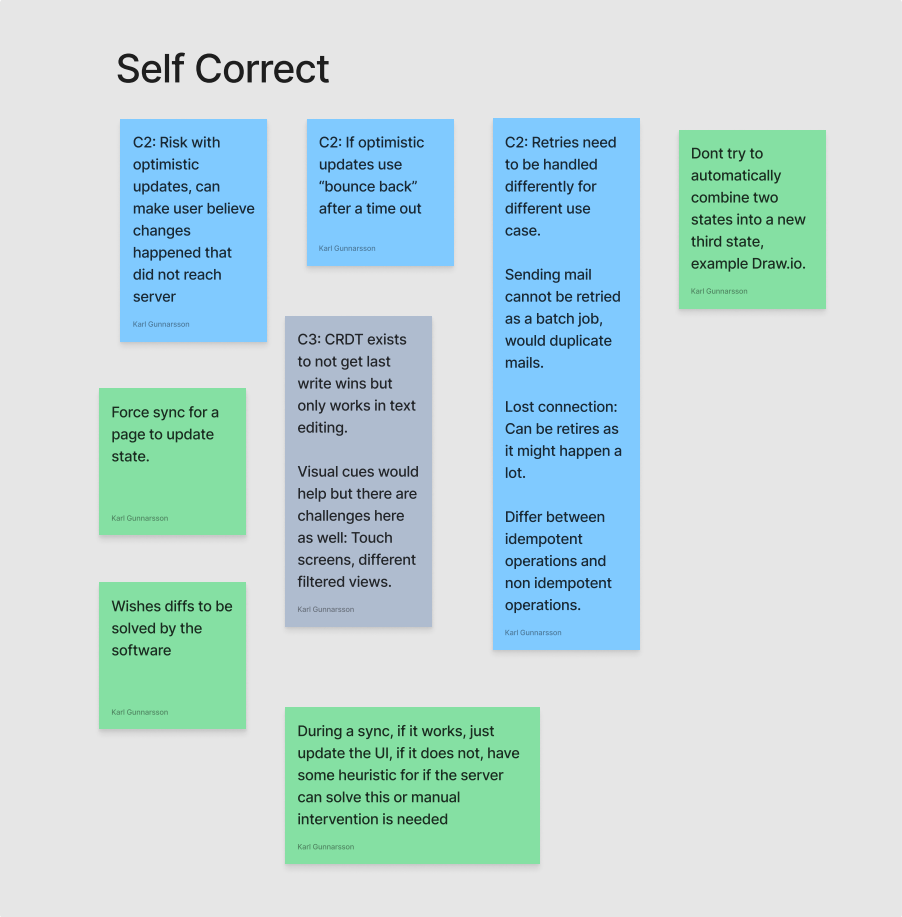

Figure A.7: Interview answers mapped onto the Self Correct strategy

Figure A.8: Interview answers mapped onto the Gag strategy.

Figure A.9: Interview answers mapped onto the Do Nothing strategy.


Visability and Communication
Important that the interface shows something has happened during a diff
Karl Gunnarsson
C1: Could use more context embedded error messages, and even possibly block actions that result in errors
Karl Gunnarsson
C1: Alterify is top small
Karl Gunnarsson
C1: Alertify works but would prefer to center it.
Karl Gunnarsson
C2: Visual indication that we are out of sync.
Karl Gunnarsson
Want: Personal history
Karl Gunnarsson
C2: Show latest saved state on server.
Karl Gunnarsson
C2: Mark if there are conflicts in the UI
Karl Gunnarsson
Visual hints for history, shadows etc.
Karl Gunnarsson
Important to know if you are out of sync
Karl Gunnarsson
Need a clear indication that there exits a conflict
Karl Gunnarsson
C2: Want optmistic updates, but also some clear indication when not in sync with server, max a few seconds
Karl Gunnarsson
C2: Building a que of pending changes and show loading indicator on sync
Karl Gunnarsson
C2: You want to know if you made a change with non updated local state
Karl Gunnarsson
C2: Old optimistic update models resulted in experienced data loss for users.
Possible fix: Transaction like pending state
Karl Gunnarsson
Important to be able to view history
Karl Gunnarsson
C3: Highlighting that new data has come into local state, transitions etc.
Karl Gunnarsson
C2: Logs could be useful, but might not work for non-technical people
Karl Gunnarsson
C2: Want a changelog for what the server will rewrite in local state
Karl Gunnarsson
Some form of overview history mapping
Karl Gunnarsson
C3: Mitigation: Show visual cues or even lock down components
Karl Gunnarsson
C3: CRDT exists to not get last write wins but only works in text editing.
Visual cues would help but there are challenges here as well: Touch screens, different filtered views.
Karl Gunnarsson
Both high level indications and low level indication
high: files
low: text rows in file
Karl Gunnarsson
C3: Visual indication on what different users are interacting with
Karl Gunnarsson
Make it clear why and where the problem occurs
Karl Gunnarsson
C3: If you have conflicts, do not want hard rejects, but instead make it clear and not intrusive what differs between states
Karl Gunnarsson
Merge diff
Karl Gunnarsson
Clear visualizations for changes and viewing diffs
Karl Gunnarsson
Want: During a sync, would prefer a diff kind of like a git merge
Karl Gunnarsson
Wants to be able to visualize what other collaborators has done / not done
Karl Gunnarsson
Minimize what the diff shows, only show impacted parts of the interface/data
Karl Gunnarsson
Want an overview how a conflicts affects my own changes
Karl Gunnarsson
Desires optimistic updates, but also desires visualization for indicating if local state is synced with server state
Karl Gunnarsson
Google docs can show text on server and text that is local
Karl Gunnarsson
Want: Preview / side by side with changes
Karl Gunnarsson
Visual hints during collaboration, cursors, highlights etc.
Karl Gunnarsson
Should be clear when an error has occurred, might not know what happened otherwise.
Karl Gunnarsson
Want: Modal, intrusiveness is fine here due to the nature of the problem
Karl Gunnarsson
Wishes for highlighting client state that might not be in sync with servers state, non intrusively.
Karl Gunnarsson