CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

UFRM: A *User Feedback Reference Model* for Managing Feedback in Dynamic Software Scenarios

A Systematic Approach to Developing and Evaluating the UFRM in Dynamic Scenarios

Master's Thesis in Computer Science and Engineering

Hoda Sheikholeslamigarvandani
Haoru Sui

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's Thesis 2025

UFRM: A *User Feedback Reference Model* for Managing Feedback in Dynamic Software Scenarios

A Systematic Approach to Developing and Evaluating UFRM in Dynamic Scenarios

Hoda Sheikholeslamigarvandani
Haoru Sui

CHALMERS
UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Chalmers University of Technology
Gothenburg, Sweden 2025

UFRM: A *User Feedback Reference Model* for Managing Feedback in Dynamic Software Scenarios
A Systematic Approach to Developing and Evaluating UFRM in Dynamic Scenarios

Hoda Sheikholeslamigarvandani and Haoru Sui

© Hoda Sheikholeslamigarvandani, 2025.
© Haoru Sui, 2025.

Supervisor: Farnaz Fotrousi, Department of Computer Science and Engineering
Examiner: Gregory Gay, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

UFRM: A *User Feedback Reference Model* for Managing Feedback in Dynamic Software Scenarios
A Systematic Approach to Developing and Evaluating UFRM in Dynamic Scenarios
Hoda Sheikholeslamigarvandani and *Haoru Sui*
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

User feedback is essential for software improvement, shaping usability, functionality, and overall user experience. However, in Dynamic Scenarios, where users try to provide feedback under high cognitive load, stress, or mental pressure due to factors such as time sensitivity, environmental uncertainty, or task-focused workflows, significant challenges arise. These situations make it difficult for both feedback senders and receivers to effectively manage the feedback process. As a result, users may delay or skip giving feedback altogether, making it harder for receivers to collect and process valuable input, especially in dynamic scenarios.
This study develops a **User Feedback Reference Model (UFRM)** to improve feedback management under such conditions. We conducted 10 semi-structured interviews with feedback receivers (such as developers and product managers) and 30 semi-structured interviews with feedback senders (end users) to understand their current issues in collecting and processing feedback in dynamic scenarios. The results were analyzed using Thematic Analysis to identify key preferences and challenges of both sides and suggest solutions. These findings were then structured into a four-layer conceptual model for managing real-time feedback in dynamic scenarios, covering scenario detection, feedback collection, processing, and response.
UFRM was validated using three real-world inspired use cases covering smart navigation systems, autonomous autopilot driving, and digital healthcare platforms to ensure its effectiveness. These use cases were derived from real feedback contexts shared by users during interviews and were applied step-by-step to test the model's adaptability and performance under realistic dynamic constraints. The findings provide insights into optimizing feedback mechanisms in dynamic scenarios, balancing the preferences of both feedback senders and receivers, and supporting better software adaptation and user experience.

Keywords: User Feedback, Dynamic Scenarios, Feedback Management, Feedback Collection, Feedback Processing, Feedback Sender, Feedback Receiver, Thematic Analysis, Use Case-Based Evaluation, UFRM.

Acknowledgements

We sincerely thank Farnaz Fotrousi, our software engineering supervisor, for her dedication and invaluable guidance throughout this thesis. Her support and encouragement have been instrumental in shaping our research. With patience and expertise, she helped us navigate the complexities of the user feedback field, providing clarity and direction at every stage. We also extend our gratitude to all interviewees who responded us, for their valuable insights and enthusiasm that enriched our study. Finally, we are deeply grateful to our families for their constant support throughout this research journey.

Hoda Sheikh & Haoru Sui, Gothenburg, May 2025

Glossary

User Feedback Information or comments provided by users about their **after interaction experience** with a software product, often used to **improve the product**.

User Refers only to **human feedback senders** in our study. The term user consistently refers to a human actor providing software feedback.

Feedback Sender An entity that provides feedback. This can be a **human user** or a **system component** (e.g., logging tool, sensor), depending on the scenario context.

Feedback Receiver An entity responsible for processing and acting on received feedback. This includes **people** like developers and support agents, or **automated feedback analysis tools**.

Explicit Feedback Feedback provided **actively** by users through direct channels (e.g., a survey form, in-app pop-up).

Implicit Feedback Feedback gathered **automatically** during user interactions, often inferred from behavior (e.g., sensor data, system logs).

Dynamic Feedback Scenario A **high-pressure situation** where users attempt to give feedback under time pressure, environmental constraints, system instability, or high cognitive load.

Static (traditional) Feedback Scenario A **calm and stable situation** where users can provide detailed feedback without time pressure or system disruption, typically **after task completion**.

Dynamic Feedback Feedback provided under our defined **dynamic feedback scenarios**. Often limited in quality or completeness due to the user's situation.

Static (traditional) Feedback Common feedback provided in **static feedback scenarios** (e.g., post-use surveys). Usually more complete and structured.

User Feedback Reference Model (UFRM) A **four-layer conceptual model** developed in this thesis to improve feedback management in dynamic scenarios by supporting detection, collection, processing, and response.

Design-Time The stage where feedback mechanisms are defined and configured **before real-time use**, based on expected dynamic conditions.

Run-Time The stage during which the system operates in **real conditions** and dynamically selects and activates suitable feedback mechanisms.

Feedback Mechanism A predefined way to collect feedback, including different properties or attributes, such as a voice message, text form, or automatic log, **chosen based on user context and scenario**.

Active Feedback Mechanism A feedback mechanism that has been **selected and activated during run-time** to match the user's current condition.

Feedback Data Package A structured group of feedback entries, including user input and contextual metadata, grouped by a certain sender and aggregated together for processing.

Feedback Processor The entity in UFRM responsible for **cleaning, categorizing, prioritizing**, and preparing feedback data for action.

Closed Feedback Loop A complete cycle in which user feedback is **collected, processed, and followed by a visible response or action**, ensuring users see the impact of their input.

Processing Feedback A systematic approach for receivers to handle user feedback, including **cleaning, categorizing, prioritizing, analyzing**, and acting on the feedback to improve a software system.

Contents

List of Figures **xv**

List of Tables **xvii**

1 Introduction **3**
1.1 Feedback Challenges in Dynamic Scenarios 3
1.2 Research Objectives and Questions 4
1.3 Research Methods 5
1.4 The User Feedback Reference Model (UFRM) 5
1.4.1 Design Goals of UFRM 6
1.4.2 Core Structure: Four-Layer Architecture 6

2 Background and Related works **7**
2.1 Introduction To User Feedback For Software Products Under Dynamic Scenario 7
2.2 Understanding The Participants In The Feedback Loop 8
2.2.1 Feedback Receivers 8
2.2.2 Feedback Senders 8
2.2.2.1 The Impact of Submission Cost and Mechanism Complexity On Engagement 9
2.2.2.2 The Impact Of Psychological Factors and Technical Level on Feedback Quality 9
2.2.2.3 The Impact of Feedback Channels on Feedback Management 10
2.3 Existing User Feedback Management Approaches 10
2.3.1 Traditional Text-Based Feedback Management 11
2.3.2 Automated Feedback Processing 12
2.3.3 Data-Driven Feedback Management 12
2.4 Main Challenges of The Existing User Feedback Management Systems . . . 13
2.4.1 Variety in Quality of Feedback Data 13
2.4.2 Difficulty in Categorizing and Prioritizing Feedback in Dynamic Scenarios 14
2.4.3 Inaccessibility of Feedback During Dynamic Conditions 14
2.4.4 Lack of Closed Feedback Loop and User Trust Mechanisms 15
2.5 Related Research Methods 15
2.5.1 Semi-Structured Interviews 15
2.5.2 Thematic Analysis and Model Conceptualization 16
2.5.3 Use Case-Based Demonstration and Evaluation 16

3 Research Methods **17**

3.1 Data Collection . . . 18
3.1.1 Selection Strategy . . . 18
3.1.1.1 Feedback Receivers . . . 18
3.1.1.2 Feedback Senders . . . 19
3.1.2 Design Interview Questions . . . 20
3.1.2.1 Feedback Receivers . . . 20
3.1.2.2 Feedback Senders . . . 20
3.1.3 Conduct Interviews . . . 20
3.1.3.1 Feedback Receivers . . . 21
3.1.3.2 Feedback Senders . . . 21
3.2 Data Analysis . . . 21
3.2.1 Step 1 - Transcription, Familiarization With the Data, and Selection of Quotations . . . 22
3.2.2 Step 2 - Selection of Keywords . . . 23
3.2.3 Step 3 - Coding . . . 23
3.2.4 Step 4 - Theme Development and Analysis . . . 23
3.2.5 Step 5 - Conceptualization of Core Layers Based on Themes . . . 24
3.2.6 Step 6 - Development of Conceptual UFRM . . . 25
3.3 Model Demonstration and Evaluation - Use Case . . . 25

4 Results **27**
4.1 From Transcripts to Codes . . . 27
4.2 Thematic Analysis with Interview Results . . . 28
4.2.1 Theme 1: Dynamic Scenarios Characteristics . . . 30
4.2.2 Theme 2: Feedback Collection in Dynamic Scenarios . . . 32
4.2.3 Theme 3: Feedback Motivation in Dynamic Scenarios . . . 35
4.2.4 Theme 4: Feedback Internal Processing and Workflow in Dynamic Scenarios . . . 37
4.2.5 Theme 5: Feedback Quality Limitations in Dynamic Scenarios . . . 40
4.2.6 Theme 6: Feedback Follow-up and Response in Dynamic Scenarios . . . 43
4.3 Insights From Thematic Analysis Results . . . 44
4.3.1 Identifying and Handling Feedback Barriers in Dynamic Situations . . . 45
4.3.2 Making Feedback Sending Easy and Timely for Feedback Senders . . . 45
4.3.3 Improving Feedback Clarity and Streamlining Internal Processing . . . 46
4.3.4 Building Trust Through Meaningful and Timely Responses . . . 47
4.4 UFRM Design . . . 47
4.4.1 Process Flow in UFRM . . . 47
4.4.2 Entity Classes in UFRM . . . 48
4.4.3 Class Calls in UFRM Layers . . . 49
4.5 UFRM Validation . . . 50
4.5.1 Demonstrate Use Case 1: Lost Navigation During a Trip . . . 50
4.5.2 Demonstrate Use Case 2: Auto-Pilot Failure on Highway . . . 52
4.5.3 Demonstrate Use Case 3: Interrupted Medical Video Call . . . 53
4.5.4 Evaluate UFRM Design Goal . . . 55

5 Discussion **57**
5.1 Key Differences Between UFRM and Traditional Feedback Management Systems . . . 57
5.1.1 Real-Time Handling Across Feedback Management Stages . . . 57
5.1.2 Automatic and Context-Driven Decision Making . . . 57

5.1.3 Sensitivity to Dynamic Scenarios . . . 58
5.1.4 Cross-Layer Integration and Adaptive Flow . . . 58
5.2 Answers to the Research Questions . . . 58
5.3 Threats to Validity . . . 61
5.4 Informed Consent . . . 62
5.5 Usage of Generative AI in This Thesis . . . 62
5.6 Future Works . . . 63
5.7 Conclusion . . . 63

Bibliography **65**

A Interviews with feedback receivers **I**
A.1 Demographic Information . . . I
A.2 User Feedback Collection . . . I
A.3 Feedback Processing & Analysis . . . II
A.4 Dynamic Scenarios . . . III
A.5 Closing Questions . . . III

B Interviews with feedback senders **V**
B.1 General Experience with Giving Feedback . . . V
B.2 Feedback in Dynamic Scenarios . . . VI
B.3 About You (Demographic Information) . . . VI

C Entity Classes and Their Associations in UFRM **VII**

List of Figures

3.1 Overview of The Whole Research Process 17
3.2 Demographic Information of Interviewed Feedback Receivers 19
3.3 Demographic Information of Interviewed Feedback Senders 20
3.4 Six Steps Conceptual Framework Development Process 22
3.5 An Example of Six-Steps Thematic Analysis to Develop a Conceptual Model 24

4.1 Word Cloud from Receivers Interview Results 27
4.2 Word Cloud from Senders Interview Results 28
4.3 From Keywords and Codes to Themes 28
4.4 Thematic Analysis Framework: Tree of Themes and Sub-Themes 29
4.5 UFRM - Process Flow Diagram 47
4.6 UFRM - UML Class Diagram 48
4.7 UFRM- Class Calls in Four Layers 49
4.8 UML Class Diagram Demonstration for Use Case 1 51
4.9 UML Class Diagram Demonstration for Use Case 2 53
4.10 UML Class Diagram Demonstration for Use Case 3 54

C.1 Entity Classes in UFRM VII
C.2 Associated Classes and Lists with Entity Classes VIII

List of Tables

1.1 Research Questions and Corresponding Objectives 5

2.1 Comparison of Existing User Feedback Management Approaches 11

subcaption

1

Introduction

User Feedback plays an important role in software engineering, offering direct insight into how software is perceived and experienced by its users. It enables software development teams to discover and resolve software defects, enhance usability, and guide the continuous evolution of systems [1] [2]. As software systems grow in complexity and user needs evolve rapidly, it becomes increasingly important to develop effective mechanisms for collecting, processing, and utilizing User Feedback [3].

In calm and stable **Static Feedback Scenarios**, users can provide clear and detailed **Static Feedback**, such as bug reports or feature requests, through formal and explicit channels such as surveys or support forms [4]. However, these traditional methods often fail in **Dynamic Feedback Scenarios**, where users face time pressure, system instability, or environmental constraints. Such scenarios commonly occur in domains like self-driving vehicles, real-time logistics systems, or emergency healthcare platforms [5].

In such high-pressure contexts, users, as **Feedback Senders**, often lack the time, have system stability constraints, or cognitive capacity to submit timely and meaningful input [6]. For example, during a sudden navigation failure in an autonomous driving system, the user may have only a few seconds to respond, which makes formal reporting impractical. As a result, valuable feedback is frequently lost or submitted too late to be actionable.

To overcome these limitations, recent studies emphasize the importance of adaptive models that can support real-time and context-sensitive feedback collection and processing in unstable environments [6]. In response, this study proposes the **User Feedback Reference Model (UFRM)**, a structured four-layer conceptual model for managing User Feedback in **Dynamic Feedback Scenarios**. UFRM is designed to enable context-aware detection, efficient collection, internal processing, and meaningful feedback responses, even when users are under time pressure or experiencing cognitive load.

1.1 Feedback Challenges in Dynamic Scenarios

Within our scope, User Feedback can generally be categorized into two types: **Static Feedback** and **Dynamic Feedback**. Static Feedback is typically provided in Static Feedback Scenarios, for example, the calm, post-task contexts where users have time and mental capacity to generate detailed and structured input [7].

In contrast, **Dynamic Feedback** emerges in Dynamic Feedback Scenarios, where we found from our interviews, the Feedback Senders usually face conditions such as **Time Pressure**, **Environmental Constraints**, **System Instability**, or **High Cognitive**

Load. These scenarios introduce significant challenges for feedback submission, internal handling, and system response. We define the following four core characteristics of Dynamic Feedback Scenarios:

- **Time Pressure**: Users face time pressure to submit feedback, such as reporting an issue or giving a quick rating, as the value of feedback decreases rapidly over time. It is required to be submitted and processed as fast as possible to ensure effectiveness. For example, when an error occurs in a navigation app while driving, the user only has a brief moment to report the problem before it becomes irrelevant.
- **Environment Constraints**: Environmental factors such as unstable network signals and physical mobility limitations hinder feedback submission. It is difficult for users to provide immediate feedback in areas with poor network conditions or in high mobility scenarios such as driving.
- **System Instability**: System instability issues, such as crashes and technical failures, can be a major challenge in dynamic feedback scenarios. It reduces user trust and makes users think that submitting feedback, such as bug reports or incident descriptions, on unreliable systems is meaningless, thus reducing their willingness to submit it. In addition, system failures can interrupt the submission process, preventing users from completing their input.
- **Task-Focused Workflows**: Users often experience high cognitive loads and high mental pressure while engaged in complex tasks, multitask, or an unstoppable process, making it difficult for them to submit feedback actively. In situations like driving, surgery, or customer service, they may delay or skip feedback, leading to missing or late responses.

These characteristics collectively result in Dynamic Feedback that is often vague, incomplete, or absent, thereby weakening the Closed Feedback Loop between sender and receiver. To address these challenges, we propose the **User Feedback Reference Model (UFRM)**, which is structured around six key **design goals**: (1) context awareness, (2) low sender effort, (3) improved feedback quality, (4) efficient organizational processing, (5) closed feedback loop, and (6) support for both feedback senders and receivers. These goals were derived from our thematic analysis of interviews and aim to address the practical limitations of existing feedback handling methods in dynamic scenarios (see `Section 1.4.1` for full details).

1.2 Research Objectives and Questions

Based on the challenges identified above and the limitations of existing feedback management methods, this research aims to design and validate the **User Feedback Reference Model (UFRM)** to better manage feedback in Dynamic Feedback Scenarios. The study is guided by the following objectives and research questions:

Table 1.1: Research Questions and Corresponding Objectives

Objectives	Corresponding RQs	Sections
OBJ1: Develop a User Feedback Reference Model (UFRM) for dynamic scenarios.	RQ1: What are the key components and characteristics of the User Feedback Reference Model (UFRM) for collecting and processing user feedback in dynamic scenarios?	4.4
OBJ1.1: Understand the main challenges and contextual characteristics influencing feedback in dynamic scenarios.	RQ1.1: What are the characteristics of dynamic scenarios?	4.2
OBJ1.2: Identify the feedback senders' preferences and challenges to ensure the model supports their needs.	RQ1.2: What are the preferences and challenges of feedback senders for providing feedback in dynamic scenarios?	4.2, 4.3
OBJ1.3: Understand feedback receivers' needs for internal processing and decision-making.	RQ1.3: What are the preferences and challenges of feedback receivers for processing user feedback in dynamic scenarios?	4.2, 4.3
OBJ2: Validate UFRM in real-world-inspired scenarios.	RQ2: To what extent is the proposed User Feedback Reference Model (UFRM) applicable in real-world dynamic scenarios?	4.5

1.3 Research Methods

This study followed **qualitative research approaches** to explore and address the challenges of user feedback in dynamic scenarios. We combined three research methods to guide the development and validation of the `User Feedback Reference Model (UFRM)`.

First, we conducted **two sets of semi-structured interviews** [8], one with feedback receivers (e.g., developers, product managers) and one with feedback senders (end users). These interviews provided in-depth information about real feedback experiences, challenges, and preferences in dynamic environments.

Second, we applied **thematic analysis** methodologies [9] and approaches designed by M. Naeem [10] to analyze the interview data and extract key themes. These themes formed the conceptual foundation for the UFRM, which we designed as a four-layer model to support real-time, context-aware feedback handling in dynamic situations.

Third, we applied a validation strategy extracted from the Design Science Research Methodology (DSRM) [11], using **use case-based demonstration and evaluation**. We tested the UFRM through three real-world-inspired use cases to simulate how the model functions across its four layers. This helped us assess whether UFRM achieves its intended design goals and effectively supports feedback management in dynamic scenarios.

1.4 The User Feedback Reference Model (UFRM)

The `User Feedback Reference Model (UFRM)` is a structured conceptual model designed to help software systems manage user feedback more effectively in dynamic scenarios. The model is flexible, supports real-time feedback handling, and works with both explicit (manual) and implicit (automatic) feedback sources. In addition to improving the feedback experience for **feedback senders**, UFRM also acts as a guideline for **feedback receivers**

to better organize, analyze, and respond to incoming feedback, even in unstable or fast-changing environments. By following UFRM, teams can make more informed design and development decisions, leading to better software adaptation and satisfaction on both sides.

1.4.1 Design Goals of UFRM

UFRM was designed with several key goals in mind, based on insights from our thematic analysis of interviews with feedback **senders** and **receivers**:

- **Context awareness**: The model should detect a dynamic situation and adapt feedback handling accordingly to the context of the **feedback sender**.
- **Low sender effort**: It should reduce the cognitive load required by the **feedback sender** to submit input, especially during stressful or time-sensitive tasks.
- **Improved feedback quality**: The model aims to reduce vague or delayed input from **senders** by capturing contextual metadata and supporting structured feedback.
- **Efficient organizational processing**: UFRM supports **feedback receivers** by enabling prioritization, classification, and internal routing to process input more effectively.
- **Closed feedback loop**: The model ensures that **feedback senders** receive meaningful and timely responses, which helps **feedback receivers** build trust and maintain user engagement.
- **Support for both senders and receivers**: UFRM is explicitly designed to support both **feedback senders** (those who provide input) and **feedback receivers** (those who process it), ensuring utility and usability for both roles.

These goals align with the research objectives presented in `Section 1.2` and form the foundation for the model structure and its validation in later chapters.

1.4.2 Core Structure: Four-Layer Architecture

UFRM is built around a four-layer architecture that reflects the full lifecycle of user feedback in dynamic scenarios. Each layer corresponds to key themes identified in our analysis and operates within the system through specific entities and processes (see `Section 4.4`):

1. **Dynamic Scenario Identification Layer**: Detects when feedback is likely to be submitted under dynamic conditions such as time pressure or system instability. It activates the appropriate mechanism based on scenario type and the context of the **sender**.

2. **Feedback Collection Layer**: Enables submission from the **sender** using context-aware mechanisms such as voice input, one-tap actions, or passive data triggers.

3. **Feedback Processing Layer**: Assists **receivers** in classifying, prioritizing, and routing the feedback to ensure it is actionable and clear.

4. **Feedback Response Layer**: Provides appropriate responses from the **receiver** back to the **sender**, maintaining a closed loop and supporting continuous improvement.

2

Background and Related works

This chapter explores the meaning of user feedback, user feedback management approach, and challenges through Literature Reviews. It provides the theoretical foundation and background support for the proposed *User Feedback Reference Model* (UFRM) by analyzing and summarizing existing research findings.

2.1 Introduction To User Feedback For Software Products Under Dynamic Scenario

User feedback refers to the opinions and experience information that **end users** convey to developers through comments, ratings, or problem reports when using **software products** [12]. This feedback not only includes specific evaluations of software functions but may also involve difficulties encountered by Senders during use or suggestions for future functions [13]. In the software development process, user feedback serves as an important bridge between users and developers, and empirical studies indicate it significantly contributes to requirements gathering and software quality improvements [14]. As software systems grow in complexity, the importance of user feedback in guiding the continuous evolution of products becomes even more pronounced.

Research further shows that user feedback profoundly impacts **optimizing user experience**. By systematically analyzing a large volume of user feedback, Receivers can uncover unmet Sender needs and even anticipate emerging requirements[15]. For example, an online education platform may find through user feedback that video loading speed has become a problem, thus optimizing server performance to improve the course experience.

The importance of user feedback is also reflected in its **guiding role in the Receivers'** decision-making [3]. User feedback offers real-world insights from the Senders' perspective, helping Receivers adjust their technical choices and prioritize features accordingly [16]. For example, in a social media application, frequent user complaints may prompt the Receiver to re-evaluate the rationality of the push algorithm. This user-centric design concept has become the basis of many successful software products.

In addition, user feedback is also considered an important support for **software market competitiveness**. Prior studies indicate that effectively using user feedback helps Receivers rapidly adapt to market needs and improve product success [14]. For example, an e-commerce platform optimized the payment process through feedback and significantly improved the user conversion rate. This dynamic process from feedback to optimization reflects its core position in the software life cycle.

2.2 Understanding The Participants In The Feedback Loop

2.2.1 Feedback Receivers

For **feedback receivers**, such as developers, product managers, and data analysts, managing user feedback is a complex and time-consuming system engineering project [17] that requires filtering non-relevant, duplicated, noisy and other useless feedback from guiding points among massive amounts of information flows, in order to optimize product features, fix technical defects, and improve the overall user experience. Currently, **manual triage** of the usability and performance of features in developed software is still a common approach adopted by many receivers [18]. Developers rely on years of experience and intuition to read comments or reports submitted by Senders one by one to determine which feedback deserves priority attention [3]. However, this method is inadequate when faced with a large user base and diverse feedback. Not only is the processing efficiency low, but it is also easily affected by personal subjective bias [19]. For example, previous studies suggest that developers focusing on backend issues may prioritise crash-related feedback and sometimes ignore interface-related suggestions, which can lead to uneven resource allocation [13] [20]. Especially in mobile applications, user feedback is often fragmented and lacks contextual information, making manual screening less effective for supporting fast development cycles [17].

To overcome the shortcomings of manual triage, **automated text analysis** has gradually become a highly respected management method. By introducing Natural Language Processing (NLP) technology, the system can classify, sentiment analyse, and topic model user feedback, thereby significantly improving processing efficiency [5]. For example, video editing software may use automated tools to identify that "long rendering time" is the most common Sender complaint, and then optimise the algorithm to shorten the processing time. In addition, this method can also discover hidden thematic patterns and provide new inspiration for product design [15]. For example, through thematic analysis, feedback Receivers may find that feedback Senders have an increasing demand for multilingual support, thereby promoting the development of international features. This technological advancement is particularly good at processing **unstructured text** (such as emotional comments or short sentences), reducing the manual burden, and providing a feasible solution for large-scale feedback management.

At the same time, implicit usage data can effectively reveal Sender behaviour patterns and potential UX issues. Tracking user clicks, browsing behaviour, and page dwell time can help developers infer usability issues and interface intuitiveness, enabling proactive UI optimisations to meet Sender expectations better [21]. For example, a fitness app may analyze that Senders spend too little time on a certain function and infer that its UI design is not intuitive enough, so they fix their UI. This combination of multi-dimensional data not only helps developers identify surface problems but also explores deep Sender behavior patterns, providing data support for product strategy adjustments.

2.2.2 Feedback Senders

The **feedback sender**, i.e., the end user, can be constrained to multiple factors when submitting feedback in dynamic software scenarios, and their motivations are significantly different from traditional static environments [22]. Studies have shown that Senders are more likely to provide feedback when they face major functional failures or highly emotional experiences, while routine or minor issues are often overlooked [23]. For example,

in a real-time navigation application, if positioning errors cause Senders to get lost, they may immediately give a complaint. Still, if the interface buttons are slightly inconvenient, Senders often choose to adapt rather than provide feedback. This selective behavior is usually prominent, such as in smart home systems, where Senders may manually adjust settings due to device response delays instead of reporting to the developer. The dynamic feedback scenarios defined in `chapter 1`, including time sensitivity, environmental uncertainty, and task-focused workflows, can further exacerbate this trend. Therefore, it is important to understand these dynamic motivations to optimise feedback collection mechanisms.

2.2.2.1 The Impact of Submission Cost and Mechanism Complexity On Engagement

In highly dynamic feedback scenarios, Senders often find it difficult to devote a lot of energy or time to writing detailed feedback reports. Therefore, cumbersome reporting processes, such as requiring manual entry of detailed information, can reduce Senders' willingness to submit feedback. For example, if mobile Senders are required to log in, upload screenshots, and provide system logs manually, they may skip submitting feedback in urgent situations. Prior studies have shown that complex feedback submission processes significantly reduce Senders' willingness to report issues[24]. In contrast, offering a one-click feedback entry or pre-filled options and automatically capturing device context can greatly shorten submission time, enabling Senders to provide feedback with minimal effort. In this way, the Receiver can obtain real faults and improvement requirements in dynamic scenarios more timely, without missing critical usage data.

At the same time, some researchers suggested combining a simple UI with an **automated capture mechanism** to improve the depth of feedback. For example, when faults such as crashes or freezes occur, modern crash-reporting tools can automatically generate detailed bug reports—including crash logs, system state, and steps to reproduce the issue. Senders only need minimal interaction to verify and submit these auto-generated reports, significantly reducing the manual effort involved [25]. In addition, if the operation log is allowed to be automatically collected in the background, it will not only reduce the Sender's operating pressure in extreme situations but also provide more complete context information for subsequent analysis. After adopting such a mechanism, the Sender's willingness to submit feedback is significantly improved even in time-sensitive dynamic scenarios (such as using software while working or performing outdoor activities). The submission cost and the complexity of the feedback mechanism have a great impact on the Sender's willingness to participate in dynamic scenarios. Combining simplified processes with automatic recording is necessary to achieve more efficient feedback collection.

2.2.2.2 The Impact Of Psychological Factors and Technical Level on Feedback Quality

The feedback behavior of Senders in dynamic situations is not only limited by time and environment, but also closely related to Senders' psychological factors and technical level [5]. When Senders have **negative expectations** about the Receivers, they may choose not to provide feedback even if they encounter serious problems, thinking that "no one will deal with it anyway." Similarly, some novices or ordinary Senders can only provide vague and scattered descriptions due to their lack of understanding of the system structure, making it difficult to accurately locate the fault or explain the key points of the require-

ments. However, high-level Senders with more professional backgrounds can provide more complete technical information or reproduction steps, which significantly improves the efficiency of subsequent debugging and improvement. Therefore, when collecting Sender opinions, we need to fully consider the differences in cognition and skills of different groups of people. We should also provide corresponding guidance and templates to avoid ignoring those technically disadvantaged groups or missing the deep insights brought by professional opinions.

In addition, the emotions and motivations of the feedback giver will also affect the **depth and accuracy** of the feedback [24]. If the Sender becomes **emotional** during an emergency, the content they submit may be overly negative and lack objective details. When there are no major pain points or benefit drivers, they tend to ignore smaller bugs or areas for improvement, thus losing opportunities for improvement. At the same time, when there are **privacy or security concerns** in the system, some Senders are unwilling to provide real account information or operation logs out of concern, resulting in a lack of key diagnostic data.

2.2.2.3 The Impact of Feedback Channels on Feedback Management

In dynamic scenarios, the design and accessibility of feedback channels play a key role in shaping Senders' willingness and ability to provide feedback. The effectiveness of feedback collection in dynamic scenarios depends largely on the usability, timing, and integration of the feedback channel into the Sender experience. Literature highlights that Senders prefer channels that are simple, fast, and embedded directly into their ongoing workflow. For instance, one-tap ratings, brief comment boxes, screenshots, or voice inputs allow Senders to quickly express their opinions without interrupting their tasks. These low-effort mechanisms are particularly important in time-sensitive or high-cognitive-load environments, such as navigating, multitasking, or interacting with critical systems. When feedback submission involves complex steps or long forms, Senders often choose to skip it, especially when under pressure or focused on other goals [26] [27].

Studies also emphasize that context-aware and well-integrated feedback channels, such as in-app pop-ups triggered by system events or errors, can improve engagement by offering relevant moments for input. Additionally, multi-modal options, such as voice messages or annotated screenshots, are more suitable for mobile and dynamic use cases than traditional typing. The PORTNEUF framework, for example, stresses the importance of proactive and continuous feedback collection, tailored to Sender experience contexts [28]. Moreover, feedback channels that provide visible outcomes, such as update logs, acknowledgments, or responses, enhance Sender motivation by reinforcing the idea that their input matters.

2.3 Existing User Feedback Management Approaches

To better explain why we propose the UFRM, it is important to first look at existing user feedback management methods and understand what they can and cannot do. Over the years, different approaches have been used to collect and handle feedback from users, including manual text reading, automated text analysis, and more recent data-driven methods that combine user input with behavior data.

The table below compares these three common approaches. It shows their main features, how they work, and what challenges they face, especially in dynamic software environ-

ments. This comparison helps to show why current methods are not enough for handling real-time, fast-changing feedback situations. These gaps form the starting point for this study, which aims to build a better model for feedback collection and response in dynamic scenarios.

Table 2.1: Comparison of Existing User Feedback Management Approaches

Criteria	**Traditional Text-Based**	**Automated Feedback Processing**	**Data-Driven Feedback Management**
Feedback Source	Explicit text (e.g., emails, app reviews)	Mostly explicit text	Explicit + Implicit (e.g., logs, sensor data)
Processing Method	Manual reading, tagging, and triage	NLP-based classification, sentiment, and topic modeling	Multi-source fusion combining user input with behavioral analytics
Timeliness	Often delayed and post-task	Faster than manual, usually offline	Supports near real-time insight with streaming data
Scalability	Low (manual effort limits coverage)	Medium to high depending on model performance	High when infrastructure is in place
Accuracy	Varies based on reviewer expertise	Affected by dataset bias, lack of context, language variation	High potential with richer context, but vulnerable to noisy signals
Limitations	Time-consuming, subjective bias, fragmented across platforms	Sensitive to training data, lacks real-time capability, poor in handling ambiguity	Requires large infrastructure, privacy concerns, difficult filtering of irrelevant data
Typical Use Scenarios	Post-task bug reporting, support forms	Sentiment flagging, issue trend detection	Real-time usage monitoring, predictive issue detection

2.3.1 Traditional Text-Based Feedback Management

Early Receivers mainly relied on **text-based** user feedback management methods, such as app store reviews, email support, user forums, and issue-tracking systems. These methods provide developers with direct channels to collect Senders' real experiences and improve software products accordingly [29]. Since this method is intuitive, easy to implement, and Senders can describe their problems and needs in free text, it has long been the core of user feedback management. However, with the evolution of software development environments, especially in dynamic software scenarios, traditional text-based feedback methods have gradually exposed multiple limitations.

First, the **lag** of feedback is a key issue. Senders usually do not submit feedback at the moment they encounter a problem, but review the problem and fill out a report after a long time interval [30]. This feedback delay makes it difficult for the Receivers to accurately restore the context of the problem, affecting the efficiency of problem location [15]. Second, the **unstructured** nature of text feedback also increases information processing difficulty. Developers often need to manually screen, classify, and analyze text submitted by Senders, which not only consumes a lot of human resources but is also easily affected by

subjective judgment, resulting in inconsistency in feedback processing [5]. Moreover, user feedback is often scattered across multiple platforms (e.g., app stores, social media, forums), forcing Receivers to integrate and compare information from diverse sources. This fragmentation greatly increases the complexity of feedback management, especially in dynamic scenarios where software changes rapidly. In such cases, the fast pace of evolution demands timely processing of feedback, which traditional text-based methods struggle to achieve, highlighting the need for shorter feedback loops in practice [31].

2.3.2 Automated Feedback Processing

With the development of machine learning and Natural Language Processing technology, automated feedback processing has gradually become an important means to improve the efficiency of user feedback management [5] [32]. These methods mainly rely on technologies such as text classification, sentiment analysis, and topic modeling to process and analyze a large amount of user feedback in an automated manner [33]. In dynamic software scenarios, automated analysis of user feedback (e.g., app reviews) can identify defect reports, feature requests, and usability issues, enabling developers to detect problems and improve the software more quickly [34]. In addition, sentiment analysis techniques can automatically flag negative feedback, allowing Receivers to address the most critical Sender complaints first [13]. Automated feedback processing further reduces manual effort and improves handling efficiency, enabling developers to respond more quickly [33]. The application of automated methods has reduced the workload of feedback management to a certain extent, improved the efficiency of feedback processing, and enabled Receivers to respond to Sender needs more quickly.

Although automated feedback processing methods have made significant progress compared to traditional methods, they still have limitations in dynamic software environments. First, these methods are highly dependent on the **quality of the training dataset**. At the same time, real-world user feedback often contains typos, slang, and ambiguities that can reduce automated analysis accuracy [33]. Second, textual user feedback often lacks sufficient context for accurate NLP interpretation. Empirical evidence from app store analyses shows user feedback frequently being brief, incomplete, or lacking clear context, making automated classification and analysis challenging [35]. In addition, many automated systems still rely on **offline processing** and cannot meet the needs of some real-time feedback management in dynamic software environments. For example, in highly interactive software systems such as smart homes and autonomous driving, developers need to be able to identify user experience problems quickly. However, existing automated techniques still struggle to support real-time feedback needs in highly interactive, time-critical systems (due to constraints like processing latency and context availability) [36]."

2.3.3 Data-Driven Feedback Management

Data-driven user feedback management methods have received increasing attention in recent years. Unlike traditional approaches or purely automated text analysis, data-driven feedback methods combine explicit feedback (actively submitted by Senders) with implicit feedback (usage behavior data) to provide richer insights [30]. Data-driven feedback loops, such as continuously collecting usage analytics alongside explicit Sender input, can generate more accurate findings. Recent continuous software engineering roadmaps highlight data-driven user feedback integration as a key area for research and practice [37]. For

instance, in autonomous driving systems, vehicle sensor data can reflect driver behavior patterns, allowing for proactive optimization by combining sensor data with user feedback [30].

Moreover, multimodal feedback fusion (e.g., combining textual reports with usage logs and screenshots) helps anticipate issues and enables developers to act proactively [30]. These methods offer richer and more precise feedback, which better supports product improvement in dynamic environments.

However, data-driven methods also face several challenges. First, **privacy protection** and **data compliance** are major obstacles. During behavior data collection, regulations like GDPR must be followed to ensure data security and legality [38]. Second, as implicit feedback often includes large amounts of **irrelevant data**, effective screening and extraction remain core research issues [39]. In addition, processing massive feedback data in real time requires **high computing resources**, which can limit deployment, especially on mobile or low-power IoT platforms [36]. Therefore, although data-driven methods offer promising directions, further research is needed in privacy, filtering, and resource optimization.

User feedback management has gradually evolved from traditional manual text-based feedback collection (e.g., app reviews, support forms), to automated NLP-based feedback processing (e.g., classification, sentiment analysis) [20], and now to more advanced data-driven approaches that integrate multiple sources such as behavioral logs, sensor data, and explicit reports [39]. Despite these improvements, each method still has some limitations in dynamic software environments [40]. Traditional methods lack real-time responsiveness, automated methods often suffer from contextual ambiguity and data bias, and data-driven methods demand high computation and strong privacy guarantees. Therefore, the UFRM proposed in this study aims to support feedback management across all key stages: collection, processing, and response. It does so by integrating multiple complementary mechanisms to overcome existing limitations and improve support for dynamic feedback scenarios.

2.4 Main Challenges of The Existing User Feedback Management Systems

2.4.1 Variety in Quality of Feedback Data

In dynamic scenarios, if Senders submit feedback in an emergency, distracted, or time-sensitive situation, the integrity and standardization of the feedback content will be seriously challenged. Some Senders will only leave a **brief and blurry description** in the interface, such as "stuck" or "cannot be turned on", but lack key contexts such as reproduction steps, device information or network environment, making it difficult for developers to locate the root cause of the fault [41].

Emotional or colloquial expressions are quite common, with spelling errors, network slang or incoherent phrases added, which reduces the accuracy of automated parsing and clustering. Especially for Receivers that intend to build machine learning models to identify problem types, if they do not plan text cleaning and noise filtering in the early stage, they will face a large amount of noise feedback in the data set, which will make it difficult for clustering results or sentiment analysis to reflect Sender pain points truly, and also to

some extent hinder the overall product evolution.

In addition, **multi-channel parallel** collection further increases the inconsistency of feedback quality. Feedback from different channels varies in format and content. For example, social media posts may be long but irrelevant, app store comments are often short and emotional, and chat-based reports are usually informal and fragmented [42]. This variation makes it hard to understand the full context of a problem and increases the effort needed to merge and compare data from multiple sources.

In dynamic software environments, these issues become more serious. Users may report the same issue through different platforms using inconsistent descriptions, making it difficult to identify duplicates or maintain consistency. Studies show that duplicate, incomplete, or low-quality feedback is common when feedback is collected from diverse sources without standardization [43]. Without proper processing, the same issue might be recorded multiple times, wasting developer resources. Therefore, improving feedback quality requires better collection methods on the front end and more reliable data cleaning and merging techniques on the back end.

2.4.2 Difficulty in Categorizing and Prioritizing Feedback in Dynamic Scenarios

As mentioned in the `section 2.3.1`, in dynamic scenarios, Receivers often receive a large amount of feedback in a variety of formats. This information often appears simultaneously in **multiple channels** such as app stores, forums, and chat tools [31]. Due to the confusion of sources, there are many duplicate and conflicting contents, and manual verification and noise elimination become particularly laborious. If we only rely on the subjective judgment of developers, we may ignore some seemingly rare but serious problems and waste energy on a large number of similar but meaningless improvement requests. Some feedback is just some simple complaints or emotional comments, and the high-quality reports that contain reproduction steps or device information are buried, making subsequent analysis more time-consuming.

In addition, Receivers often lack **automated classification** and **merging strategies**, making it difficult to aggregate similar faults together in a timely manner [28]. Even if there are scripts that can identify keywords and make preliminary groupings, there will still be many missed detections or misjudgments due to semantic ambiguity or spelling errors, so core bugs or requirements will not receive due attention. At the same time, developers need to balance the different demands for basic defect repair and new feature launches. Even if some bugs are found to have a large impact, they may be temporarily shelved due to resource shortages, causing Senders to have negative emotions and question the efficiency of the Receiver.

2.4.3 Inaccessibility of Feedback During Dynamic Conditions

In dynamic environments, feedback senders may encounter **technical issues**, such as unstable systems or poor connectivity, or **environment constraints**, such as the lack of good network connectivity in their environment. These factors may discourage them from submitting feedback even if they are motivated to do so [44]. In these cases, traditional feedback systems are often not applicable because they are usually designed for stable desktop environments and lack support for timely or context-aware input [45]. When feedback tools do not integrate smoothly into the sender's workflow, the receiver may lose

the opportunity to make meaningful reports. If feedback cannot be submitted quickly and with minimal disruption, many senders will abandon the intention of providing feedback [46].

2.4.4 Lack of Closed Feedback Loop and User Trust Mechanisms

Some feedback systems do not offer a clear feedback loop that informs **Senders** how their input is processed or whether it led to any action. Research shows that **visible and timely responses**, such as **status messages**, **public updates**, or **simple acknowledgments**, help maintain user engagement and increase trust in the system [45]. However, many systems still lack these features or rely only on automatic email replies, which are often overlooked. In **dynamic environments**, where feedback is usually submitted under urgent needs, users expect faster confirmation and clearer outcomes. Without confirmation or follow-up, feedback senders may feel uncertain about the value of their contribution, which can reduce trust and lower their willingness to provide feedback again [6].

2.5 Related Research Methods

In this section, we review three core research methods that formed the basis of our study: semi-structured interviews, thematic analysis, and use case-based demonstration and evaluation. These methods were selected after a review of key literature that supports their use in qualitative, exploratory, and design-oriented research.

2.5.1 Semi-Structured Interviews

Semi-structured interviews are a widely used **qualitative data collection** method that combines the consistency of structured interviews with the flexibility of open conversations. According to the SAGE Encyclopedia of Social Science Research Methods [8], this method involves pre-determined guiding questions while still allowing space for follow-up questions and exploration of emerging topics. This balance makes semi-structured interviews particularly effective for studying complex human behaviors, attitudes, and perceptions. A key strength of this method lies in its adaptability across disciplines such as education, healthcare, and information systems. It enables researchers to maintain a consistent framework while still capturing diverse and unanticipated insights from participants. Unlike structured interviews, which may limit the depth of responses, semi-structured interviews invite elaboration and storytelling, which often reveal nuanced motivations or problems that researchers did not initially anticipate.

Palinkas et al. [47] emphasize the importance of purposeful selection when conducting semi-structured interviews in applied research contexts. This selection strategy focuses on selecting information-rich cases, individuals who are especially knowledgeable about the topic. The goal is not statistical generalizability but rather in-depth understanding.

In addition, Fusch and Ness [48] introduce the concept of **data saturation**, a critical benchmark in qualitative interviewing. Data saturation occurs when no new themes or insights emerge from continued interviews. Achieving saturation ensures data adequacy and supports the trustworthiness of qualitative findings.

2.5.2 Thematic Analysis and Model Conceptualization

Thematic analysis is a foundational technique in **qualitative research**, used for identifying, analyzing, and interpreting patterns or "themes" within qualitative data. Braun and Clarke [9] define it as a flexible yet structured approach that allows researchers to move from unorganized text to meaningful insights. This method is not bound to any particular theoretical framework, which makes it accessible for researchers from different backgrounds and disciplines.

In our study, thematic analysis was used to analyze interview data from both feedback senders and receivers. We followed the first four steps in the six-step model proposed by Naeem et al. [10] for conducting thematic analysis, including: (1) transcription of data, (2) keyword identification, (3) systematic coding, and (4) grouping of codes into themes. These steps helped us identify patterns in the data and organize them into coherent themes. This combined approach supports both **inductive** and **deductive** reasoning [9]. In the inductive mode, themes emerge directly from the data without a predefined coding structure. In contrast, the deductive approach uses a coding frame informed by theory or existing literature.

Steps 5 and 6 in Naeem et al.'s model go beyond traditional thematic analysis and focus on conceptual model development. In our study, we used these additional steps to interpret themes in relation to the research questions and to construct the UFRM framework. Both sources emphasize that thematic analysis is especially suitable for exploring how people make sense of their experiences, particularly in under-researched or evolving domains. It transforms qualitative narratives into structured insights while retaining the richness of the original data.

2.5.3 Use Case-Based Demonstration and Evaluation

The **Design Science Research Methodology (DSRM)** developed by Peffers et al. [11] provides a structured approach to designing and validating artifacts in information systems research. One of the core principles of DSRM is that any proposed solution must be evaluated for its effectiveness, utility, and applicability. The authors outline a six-step process: problem identification, objective definition, design and development, demonstration, evaluation, and communication.

The **demonstration and evaluation phases** are particularly relevant for conceptual model validation. During the demonstration phase, the artifact, such as a framework, process, or diagram, is applied to a practical use case or scenario to show how it functions in context. This approach does not require real-world deployment or user testing but instead uses realistic, well-defined examples to simulate application. Prefers et al. refer to these as **use case applications**, which are ideal for demonstrating the potential of an artifact in a safe, controlled environment.

The subsequent evaluation phase involves assessing whether the artifact meets its intended design goals. This can include measures such as usefulness, accuracy, completeness, and alignment with stakeholder needs. Evaluation methods can vary, ranging from interviews and observations to comparisons against benchmarks, but the goal is always to assess how well the solution solves the original problem.

3

Research Methods

To answer the RQs listed in `Section 1.2`, this chapter shows the research methods employed to create and validate the **User Feedback Reference Model (UFRM)** for managing user feedback in dynamic scenarios. It is structured into three key sections:

- `3.1 Data Collection`: details the interview process.
- `3.2 Data Analysis`: covers thematic analysis and model construction.
- `3.3 Model Demonstration and Evaluation`: presents the validation of UFRM.

An overview of the research steps and their relation to the research objectives and questions is illustrated in `Figure 3.1`.

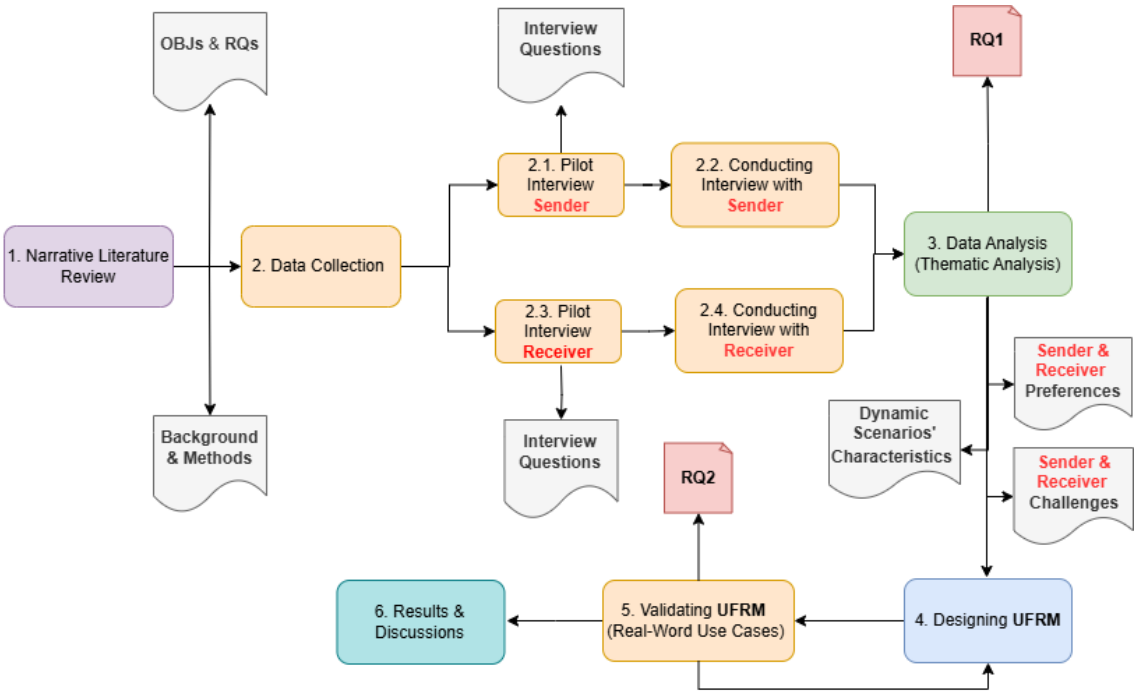

Figure 3.1: Overview of The Whole Research Process

3.1 Data Collection

This section describes in detail the implementation process of **data collection** in our study. We aim to explore user feedback management in dynamic scenarios through **semi-structured interviews** [8] to provide an empirical basis for the **development** of the **UFRM**. Our data collection focuses on two key participants: **Feedback Receivers** (such as developers and product managers) and **Feedback Senders** (i.e., end users), representing the supply and request perspectives of feedback management.

We adopted a **purposive selecting** technique [47] and recruited **10 Feedback Receivers** and **30 Feedback Senders**. Considering the difficulty of recruitment and interviews, we first interviewed Receivers, which also provided preliminary insights for designing and implementing questions for interviewing Senders.

3.1.1 Selection Strategy

This section explains how we selected participants using purposive sampling strategies for two groups: **Feedback Receivers** and **Feedback Senders**. Our goal was to ensure relevance to dynamic software scenarios, participant diversity, and thematic saturation.

3.1.1.1 Feedback Receivers

Feedback Receivers were selected from professionals with experience in managing user feedback systems in time-sensitive and complex environments. We contacted over 100 candidates through professional networks (e.g., LinkedIn, forums) and enterprise partnerships, and conducted interviews with **10** qualified participants between February 5 and March 5, 2025.

The participants represented five industry sectors: 50% worked in automotive and autonomous driving, while the remaining came from logistics, navigation, telecom, and medical software. Their roles included product managers (5), developers (4), and a data analyst (1). Most had more than 10 years of experience and were advanced in processing feedback (`Figure 3.2`).

- **Feedback Experience:** Selection prioritized professionals familiar with fast-changing systems such as real-time navigation and emergency coordination. This aligns with the study's goal to understand feedback in dynamic environments (OBJ1).
- **Data Saturation:** Following qualitative research standards [48], new insights began to repeat after the 8th interview, and saturation was confirmed at the 10th.
- **Anonymity:** Participants' voices were anonymized using codes (e.g., R01–R10) and securely stored for research use only.

Recruitment was challenged by data privacy concerns and limited availability. Flexible scheduling and secure interview formats (e.g., using Zoom meetings) were used to address this.

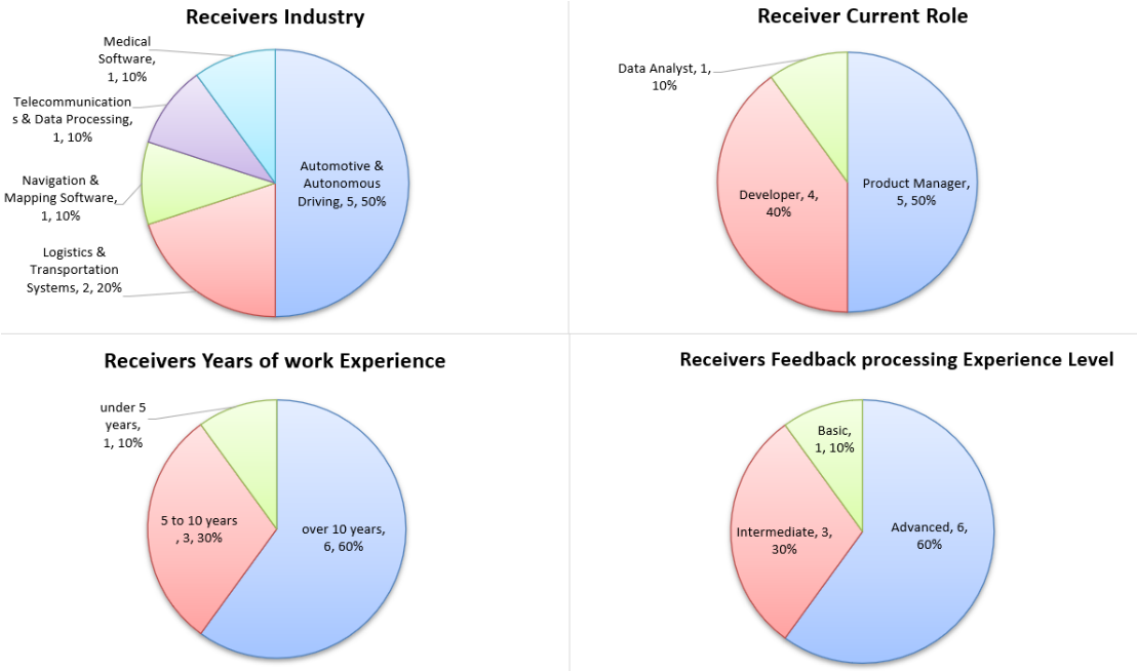

Figure 3.2: Demographic Information of Interviewed Feedback Receivers

3.1.1.2 Feedback Senders

We recruited **30** participants between March 10 and March 29, 2025, using open calls via social media, university mailing lists, and personal networks. Senders were selected based on experience using feedback features in mobile or web apps under dynamic conditions such as navigation, e-commerce, or emergency services.

As shown in `Figure 3.3`, the participants varied across application contexts: 43% shared e-commerce experience, while others came from education, social media, transport, and medical apps. Most were aged 30-50, with a majority being female. Their digital proficiency ranged from beginner to highly proficient.

- **Feedback Experience:** Participants who had previously given feedback under time pressure or unstable conditions were prioritized. These conditions reflect typical dynamic scenarios the study aims to support (OBJ1).
- **Data Saturation:** After 30 interviews, no major new insights emerged, confirming that thematic saturation was reached [48].
- **Anonymity:** All responses were anonymized using sender IDs (S01–S30), with audio stored securely for transcription and coding.

Although recruitment was easier than for Receivers, we still addressed some concerns about data use and scheduling by offering flexible online formats and informed consent procedures.

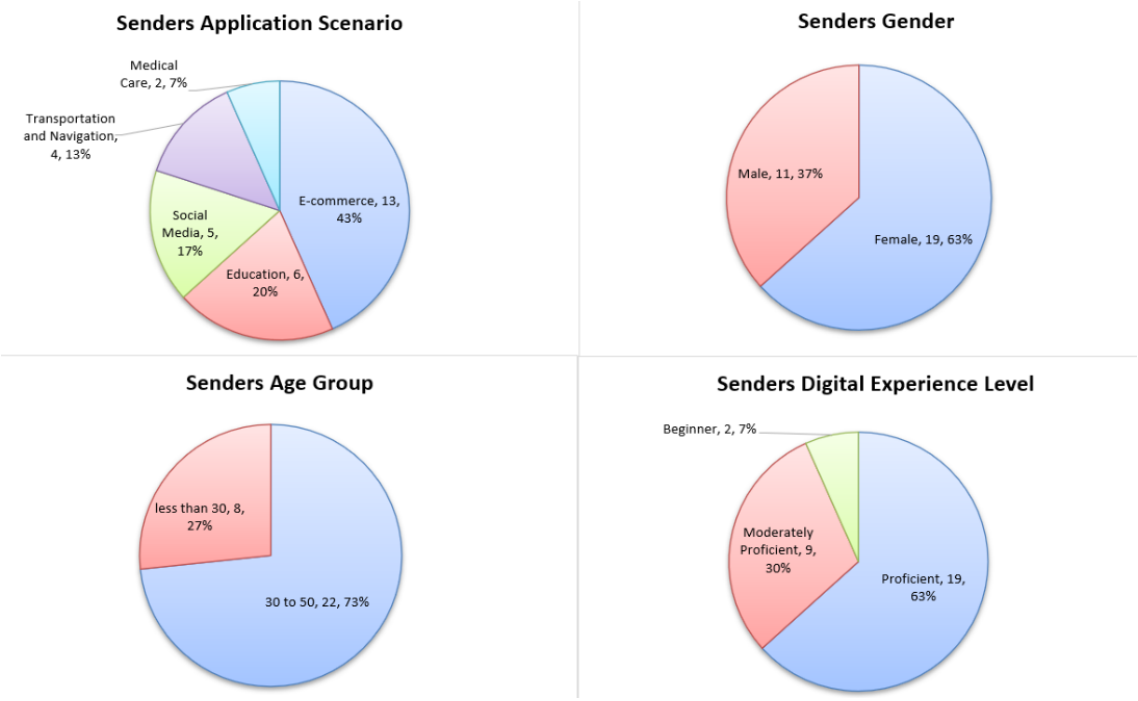

Figure 3.3: Demographic Information of Interviewed Feedback Senders

3.1.2 Design Interview Questions

3.1.2.1 Feedback Receivers

In order to reach out all our goals in OBJ1, we designed **24 questions** for **Feedback Receivers**, divided them into **5 parts** (see Appendix A), focusing on asking about challenges of Receivers in feedback collection, processing, analysis and their special needs, their preferences in dynamic scenarios and improvement suggestions. We tried to design the questions as simple as possible and without overlaps to be understandable for all levels of proficiency in interviewees and avoid their frustrations.

3.1.2.2 Feedback Senders

Aiming to answer all our research questions in RQ1 and achieve our research objective OBJ1, to understand challenges and preferences of feedback Senders in Dynamic Scenarios, we designed **14 questions** for **Feedback Senders**, divided them into **3 parts** (see Appendix B), covering the general feedback experience, Motivation and preventions, behavior of dynamic scenarios, and improvement suggestions. We tried to design non-redundant questions with minimum complexity to be understandable for all diverse interviewees.

3.1.3 Conduct Interviews

Before all the formal interviews, we conducted **pilot studies** with our supervisor and two of our volunteer friends with relevant experience, to evaluate the clarity, relevance, and rationality of the structure of the interview questions. This is a key step in qualitative research to ensure that the questions were scientifically designed, easy to understand, and can effectively promote good responses. Through pilot studies, we aimed to identify and modify questions that may cause ambiguity or appear redundant to avoid confusion

among participants when answering. We also aimed to verify whether these questions can effectively cover the research objectives and ensure that the interview content is consistent with the research direction. In addition, we also paid attention to the impact of the number and complexity of questions on the cognitive burden of participants to prevent them from feeling fatigued during the interview. We confirmed that all questions could be understood and answered smoothly by the interviewees without obvious obstacles. The results of the pilot interview showed that some questions needed to be adjusted, such as by simplifying the wording and optimizing the order of questions to improve overall clarity and fluency. After these modifications, the interview guide was more concise, clearly structured, and easy to answer for the subsequent formal interviews. When conducting interviews, all interviews were recorded, transcribed using Microsoft Word 365's automatic tool, manually verified, and stored anonymously.

3.1.3.1 Feedback Receivers

We interviewed **10 Feedback Receivers**, with each session lasting **60-80 minutes**, conducted online (via Zoom) or offline (at Chalmers University). Before starting, we explained the study's purpose, obtained informed consent, and assured anonymity. One researcher led the interview, while another recorded audio and took notes. We used guiding questions but encouraged free expression to capture detailed insights.

3.1.3.2 Feedback Senders

We interviewed **30 Feedback Senders**, with each session lasting **5-15 minutes**. Due to the simpler questions, we used both **formal interviews** (semi-structured questions, full recordings) and **informal interviews** (flexible questions, brief notes). Interviews were conducted via Zoom, phone calls, or in person to fit participants' schedules. To avoid bias from leading questions, we increased the number of interviews to gather more data, ensuring richer insights.

3.2 Data Analysis

We used **Thematic Analysis** as the main method to analyze our **interview results** from the receivers and senders. This approach is widely used for identifying, analyzing, and reporting patterns (themes) in **qualitative data** [9]. It offers a flexible but structured way to interpret rich Sender input and is especially suitable for understanding complex topics like user feedback management in dynamic software scenarios.

We followed the **Process of Thematic Analysis to develop a Conceptual Model** in qualitative research as described by Naeem et al. [10] to support the development of our conceptual model. This process helps turn interview data into clear themes and insights. `Figure 3.4` shows the steps we used to organize and analyze the data, which later helped us build the UFRM framework.

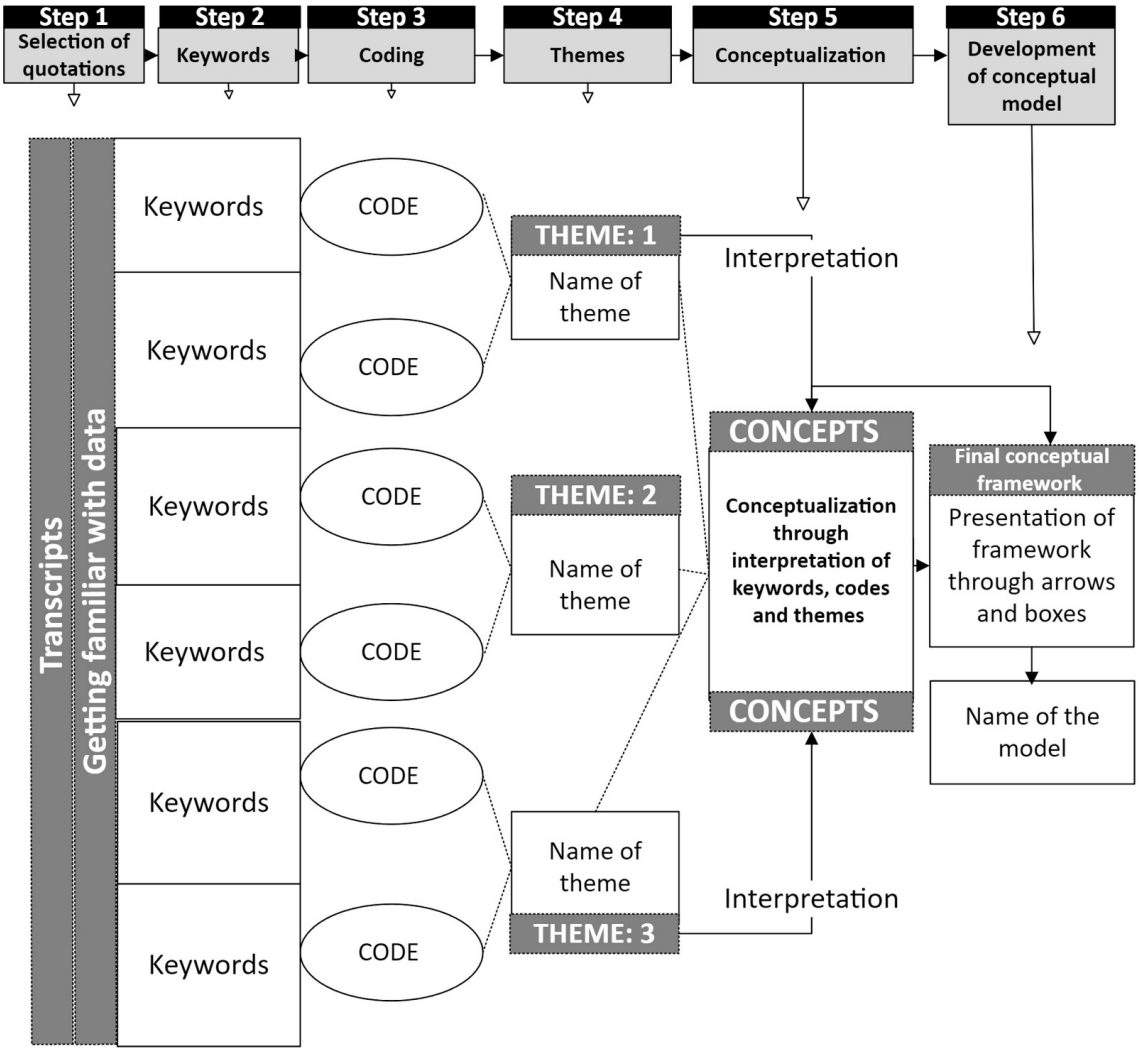

Figure 3.4: Six Steps Conceptual Framework Development Process

We applied `Steps 1-3` of the thematic analysis process separately to the interview data collected from **Feedback Senders** and **Feedback Receivers**. This allowed us to capture distinct perspectives from each group during transcription, keyword selection, and coding. Then, we combined the results from both groups to perform `Steps 4-6`, integrating the insights to develop themes, conceptualize relationships, and build the UFRM framework. We first present a detailed textual explanation of the six steps used in our thematic analysis below. And there is `Figure 3.5`, which is presented later and shows how we conduct the six-step thematic analysis in our research process.

3.2.1 Step 1 - Transcription, Familiarization With the Data, and Selection of Quotations

In the first step, we transcribed the interview recordings into **text format** using an automatic transcription tool from **Microsoft Word 365**, due to its high accuracy and seamless integration with the text processing workflow, which reduced post-editing time and was easily accessible through our university account. Then, we made manual corrections to ensure accuracy. To protect participants' **privacy** and comply with ethical

standards, we removed personal identifiers (e.g., names, company details) and eliminated off-topic or distracting content. We also **manually normalized** the data by extracting key concepts, removing duplicates, and standardizing expressions to ensure consistency.

Next, we read each transcript several times to become familiar with the content and context. Through this repeated reading, we noted emerging patterns, such as recurring phrases related to **time pressure** or **task urgency** in dynamic environments. We highlighted segments that contained meaningful insights or illustrated challenges relevant to feedback under pressure. During this process, we selected **representative quotes** that aligned with our research objectives. These quotations were chosen based on their clarity, relevance to dynamic feedback scenarios, and ability to reflect key issues expressed by participants. For example, quotes describing in-task interruptions, stressful use conditions, or urgent decision-making were prioritized to ensure the themes were grounded in concrete user experience.

3.2.2 Step 2 - Selection of Keywords

In the second step, we **manually** extracted **keywords** from the transcripts into Excel, to capture core concepts and experiences of participants in dynamic scenarios. We reviewed the data to identify recurring terms and meaningful patterns, such as **time pressure**, **urgency**, and **fast**, which feedback senders frequently mentioned due to their situational importance [10]. Keywords were selected based on their frequency, relevance to research objectives, and ability to reflect real experiences of participants, ensuring that they were derived directly from the data. This process prepared the data for the next coding stage.

3.2.3 Step 3 - Coding

In the third step, we coded the data by assigning concise labels to summarize the codes, transforming words into actionable units for analysis. The Coding was performed **manually** in Excel, with iterative refinement between the two researchers until consensus was reached. Using a combination of **inductive** (data-driven) and **deductive** (research-driven) methods, we created codes like **time constraint** and **environmental limitation** to capture key feedback management issues in dynamic scenarios. These codes were derived from the **keywords** identified earlier, ensuring alignment with research objectives. As shown in `Figure 3.5`, keywords such as “time pressure” and “bad signal” were grouped into codes like “Time Constraint” and “Network Instability,” which later contributed to themes like “Dynamic Scenarios Characteristics.” We then clustered related codes to explore connections, preparing for theme development in the next step.

For Steps 1-3, some simple examples of how we find keywords from word clouds, and how keywords and codes become themes, can be seen in the Chapter 4. Results - `Section 4.1`.

3.2.4 Step 4 - Theme Development and Analysis

The fourth step is to integrate the codes into **themes** and identify **patterns and relations** in the data to answer the research questions. **Themes** were **higher-level summaries** of the **codes**, which not only reflect the extracted elements of the data, but also reflect meaningful explanatory concepts. After completing the keyword extraction and coding work, we summarized the codes with similar meanings and summarized representative **themes** and their **sub-themes** based on repeated comparison and verification.

The theme identification process was guided by the research questions and combined with the patterns, views, and situations that recurred in the participant interviews. For example, several interviewees mentioned that it was difficult to provide effective feedback when "time was tight" and "environment was complex". We classified this type of coding under the theme of "Contextual Challenges" and further refined it into sub-themes such as "Time Pressure" and "Environment and System Instability". The entire development process was based on **Inductive logic (inductive approach)**, supplemented by some theoretical guidance to ensure that the theme was both in line with the data and theoretically meaningful [9].

So far, we have identified **6 main themes**, which cover the key aspects of user feedback behavior and reflect main challenges and preferences of feedback senders and receivers in **Dynamic Scenarios**.

The detailed results of step 4 can be seen in the Chapter 4. Results - `Section 4.2`.

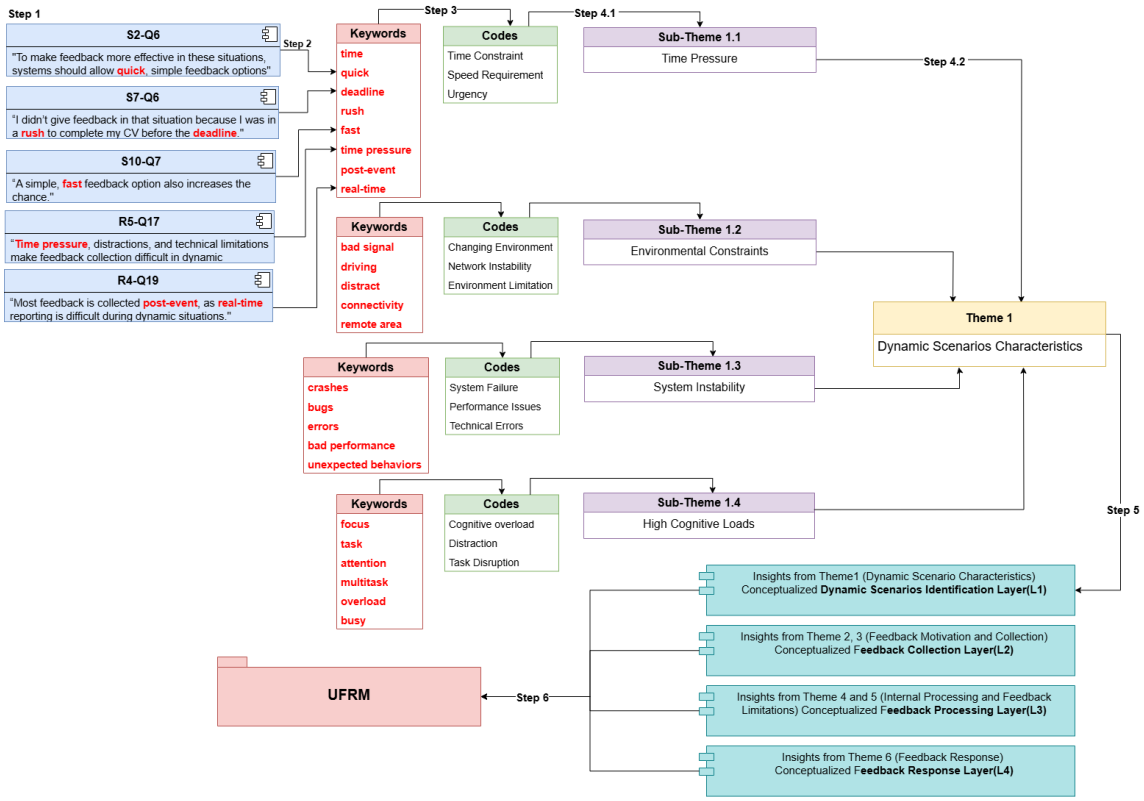

Figure 3.5: An Example of Six-Steps Thematic Analysis to Develop a Conceptual Model

3.2.5 Step 5 - Conceptualization of Core Layers Based on Themes

After completing the thematic analysis, we proceeded to the fifth step: **conceptualization**. This step transforms the six identified themes and their sub-themes into a higher-level structure that reflects the user feedback lifecycle in dynamic scenarios. By comparing and interpreting the themes, we found that they naturally correspond to four functional stages of feedback processing. These four stages formed the conceptual backbone of the UFRM.

Theme 1 (Dynamic Scenario Characteristics) became the **Dynamic Scenario Identification Layer**, which captures external and psychological conditions under which feedback is generated.

Themes 2 and 3 (Feedback Motivation and Collection) were grouped into the **Feedback Collection Layer**, highlighting Sender drivers and methods for submitting feedback.

Themes 4 and 5 (Internal Processing and Feedback Limitations) informed the **Feedback Processing Layer**, focusing on how feedback is classified and handled internally.

Theme 6 (Feedback Response) shaped the **Feedback Response Layer**, ensuring user feedback is acknowledged and closed.

The results of Step 5 are discussed in `Section 4.3` represent the output of this conceptualization step. They show how user experiences and challenges were translated into design ideas, which directly support the construction of the User Feedback Reference Model.

3.2.6 Step 6 - Development of Conceptual UFRM

With the conceptual structure established, we moved to the sixth step: building the actual model UFRM. The UFRM integrates the findings from Step 5 into a complete design artifact.

To operationalize the UFRM, we developed two key components:

- A **Process Flow Diagram** that illustrates how feedback flows across four layers (see `Section 4.4.1`).
- A **UML Class Diagram** that defines the data structure, entities, and their relationships (see `Section 4.4.2`).

These two parts turn our themes into a clear and practical model. The process diagram shows how feedback moves from detection to response, while the class model ensures each entity is explicitly defined.

The detailed results of step 6 can be seen in the Chapter 4. Results - `Section 4.4`.

3.3 Model Demonstration and Evaluation - Use Case

To evaluate the Validity and applicability of the proposed model (UFRM), we adopted one of the recommended evaluation approaches from the **Design Science Research Methodology (DSRM)** proposed by Peffers et al [11]. According to their framework, validating design artifacts can be carried out in two complementary stages:

- **Demonstration**: showing how the model functions in realistic situations by applying it to practical use cases.
- **Evaluation**: assessing whether the model effectively addresses the original problem and meets its intended design goals.

According to this approach, we validated our model using three real-world-inspired **use cases**: smart navigation, autonomous driving, and digital healthcare. These use cases

show typical dynamic situations where Senders need to give feedback quickly but often find it hard.

For the demonstration stage, we applied our two core model artifacts, the UML Class Diagram and the Process Flow Diagram (`see Section 4.3.1`) to each use case. We mapped each scenario step-by-step to the four layers of UFRM: (L1) detecting the feedback situation, (L2) collecting feedback, (L3) processing it, and (L4) generating responses. This helped us visualize how the model works in action, under time pressure or system instability.

In the evaluation stage, we assessed whether the model achieved its design objectives, such as enabling quick, easy, and adaptive feedback interactions in dynamic conditions. These evaluation points were based on our research goals defined in `Section 1.4`. The results of this validation were discussed further in `Section 4.4` and reflected on in `Chapter5`.

4

Results

4.1 From Transcripts to Codes

This section describes how the interview data were prepared and structured for analysis. Following the six-step thematic analysis and conceptual model design approach [10], we first worked on **Steps 1-3** mentioned in the `Section 3.2` through transcript familiarization, open coding, and initial theme generation as an integrated process. This phase provided the analytical foundation for the thematic structure of our study and laid the groundwork for the conceptualization of the UFRM.

We began by transcribing all interview recordings from both feedback senders and receivers. The transcripts were carefully read and annotated, allowing us to identify recurring ideas and expressions relevant to feedback in dynamic software scenarios. We then **manually extracted and labeled** meaningful content segments based on frequently mentioned terms, participant phrases, and topic relevance. Each code was iteratively refined by grouping semantically similar ideas together, combining participant terms and researcher-generated concepts. It was explained in details in `Section 3.2 (3.2.1 to 3.2.3)` that how we manually performed codes from transcripts.

To support the open coding process, we used word cloud visualizations as an auxiliary tool for identifying frequently mentioned terms in the interview transcripts. The visualized patterns provided a preliminary overview of common expressions used by feedback Senders and Receivers. These results helped guide the manual coding process by highlighting candidate terms and recurring linguistic patterns. `Figure 4.1` and `Figure 4.2` show the word clouds for each group.

Figure 4.1: Word Cloud from Receivers Interview Results

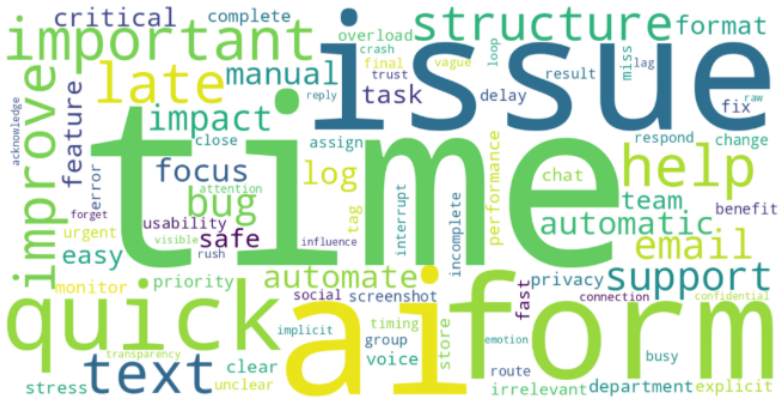

Figure 4.2: Word Cloud from Senders Interview Results

After coding, we organized the codes into larger themes based on meaning. We reviewed and refined the themes to make sure they were clear, consistent, and relevant to the study. These themes later shaped the structure of the UFRM model. `Figure 4.3` shows how we moved from raw keywords and codes to final themes.

Label	UFRM Layer	Theme	Sub-theme	Codes	Keywords
Number of items	4	6	23	56	84
UFRM structure	L1: Dynamic Scenarios Identification Layer	T1: Dynamic Scenarios Characteristics	T1.1 : Time Pressure	Time Constraint (152), Speed Requirement (152), Urgency (152)	time (372), rush (3), deadline (5), quick (186), fast (25)
			T1.2: Environment and System Instability	System Failure (3), Network Instability (3), Environment Limitation (3)	connection (3), crash (3), unstable (5), signal (5), lag (3)
			T1.3: High Cognitive Load and Task disruption	Cognitive Overload (33), Distraction (33), Task Interruption (33)	focus (42), task (36), attention (3), multitask (5), overload (12), busy (6)
	L2: Feedback Collection Layer	T2: Feedback Collection in Dynamic Scenarios	T2.1: Explicit and Implicit Collection Strategies	Explicit Input (41), Passive Logging (41), Behavior Monitoring (41)	explicit (15), implicit (3), automatic (54), manual (40), monitor (29)
			T2.2: Typical Feedback Topics	Bug Report (128), Feature Request (128), Usability Feedback (128)	bug (63), error (12), issue (240), feature (36), usability (12), performance (21)
			T2.3: Multimodal Feedback Formats	Visual Feedback (50), Audio Feedback (50), Written Input (50)	text (111), voice (30), screenshot (9), screen (5), picture (5)
			T2.4: Variety of Feedback Channels	Support Tickets (129), Social Media (129), App Review (129)	email (63), chat (27), support (66), form (204), store (21), social (6)
		T3: Feedback Motivation in Dynamic Scenarios	T3.1: Reasons for Giving Feedback	Problem Resolution (100), Contribution Desire (100), Perceived Impact (100)	help (114), improve (81), fix (18), benefit (9), important (140)
			T3.2: Importance of Timing and Mood	Mood Sensitivity (15), Feedback Timing (15)	timing (6), mood (5), emotion (2), stress (22)
			T3.3: Ease, speed, and non-disruptiveness of feedback	Non-Disruptive Process (56), Ease of Use (56)	easy (40), fast (25), interrupt (6), quick (186)
			T3.4: Influence of Trust and Privacy on Giving Feedback	Trust Factor (34), Privacy Concerns (34)	trust (4), privacy (24), safe (38), confidential (2)
	L3: Feedback Processing Layer	T4: Feedback Internal Processing and Workflow in Dynamic Scenarios	T4.1: Manual And Automated Feedback Processing	Manual Processing (192), Automated Workflow (192)	automate (52), manual (40), ai (272), log (42), monitor (29)
			T4.2: Feedback Categorization Mechanisms	Feedback Sorting (11), Tagging System (11)	category (5), classify (5), tag (14), group (8)
			T4.3: Feedback Prioritization	High Priority (65), Issue Severity (65)	priority (18), important (140), critical (40), urgent (10)
			T4.4: Feedback Routing and Responsibility Assignment	Department Routing (45), Ownership Assignment (45)	assign (10), team (40), department (34), route (6)
		T5: Feedback Quality and Structure Limitations in Dynamic Scenarios	T5.1: Unclear and Incomplete Feedback	Lack of Clarity (10), Incomplete Message (10)	unclear (8), incomplete (8), vague (4), confuse (5)
			T5.2: Irrelevant, Redundant, or Inaccurate Feedback	Noise Feedback (5), Redundancy (5), Inaccuracy (5)	irrelevant (15), repeat (5), duplicate (5), wrong (5)
			T5.3: Unstructured or Hard-to-Process Feedback	Unstructured Data (58), Formatting Issues (58)	structure (76), format (38), organize (5), raw (2)
			T5.4: Delayed or Missed Feedback	Missed Opportunity (58), Feedback Delay (58)	delay (12), late (92), miss (10), forget (2)
	L4: Feedback Response Layer	T6: Feedback Follow-up and Response in Dynamic Scenarios	T6.1: Timely Responses to Feedback	Acknowledgment (38), Fast Response (38)	respond (8), reply (2), acknowledge (2), quick (186)
			T6.2: Communication Transparency	Open Communication (16), Honest Disclosure (16)	transparency (2), clear (28), honest (5), visible (2)
			T6.3: Feedback Loop Closure	Feedback Completion (20), Loop Closure (20)	close (8), complete (22), loop (4), final (6)
			T6.4: Visibility of Feedback Impact	Feedback Effect (37), Visible Change (37)	impact (54), change (10), influence (2), result (8)

Figure 4.3: From Keywords and Codes to Themes

Related codes were then grouped into larger thematic patterns based on their semantic meaning. These initial themes were refined to ensure coherence and coverage, forming the basis for our model design.

4.2 Thematic Analysis with Interview Results

To identify and structure the core insights from our studies, in **Step 4 - Theme Development and Analysis**, we clustered the codes into sub-themes and major themes that

reflected the challenges and preferences that emerged when handling feedback in dynamic scenarios. We refined our findings into **6 major themes** and **24 sub-themes** through this process, then identified **4 layers** based on these themes to describe the feedback processing stages, as shown in `Figure 4.4`.

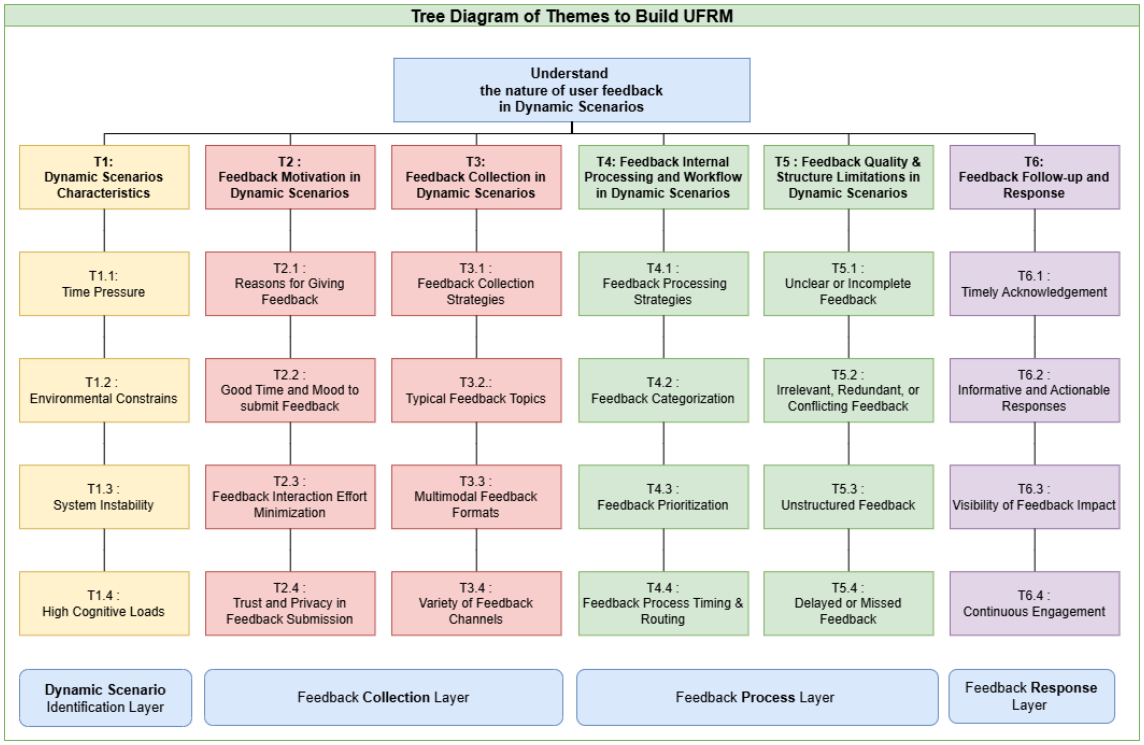

Figure 4.4: Thematic Analysis Framework: Tree of Themes and Sub-Themes

Layer 1: Dynamic Scenario Identification

This foundational layer originates from **Theme 1** (Dynamic Scenarios Characteristics), which emphasizes external and internal conditions like time pressure, system instability, and cognitive load that impact feedback behavior. This layer supports the system's ability to recognize when feedback is happening under dynamic constraints.

Layer 2: Feedback Collection

Derived from **Themes 2** and **3** (Feedback Collection Methods and Feedback Motivation), this layer explains how feedback is initiated, including the consideration of trigger timing, data formats, user factors, etc. The mechanism for capturing Sender input is adapted to Senders' states and environmental factors, combining both explicit and implicit feedback collection methods.

Layer 3: Feedback Processing

Themes 4 and **5** (Internal Processing and Quality Limitations) informed the third layer, which handles the user feedback by classification, prioritization, routing, etc. It ensures that receivers establish efficient workflows that balance automation and manual review of dynamic cases.

Layer 4: Feedback Response

Theme 6 (Feedback Follow-up and Response) defines the last layer, where the system closes the loop by responding to Senders. This includes personalized responses, transparency of feedback impact, and continued engagement, all of which are crucial in dynamic contexts to maintain trust and motivation for Senders.

In the following subsections, we present real examples of original **quotes** from interviews with Senders (e.g., S1, S2) and Receivers (e.g., R1, R2) to illustrate how we extracted **sub-themes and themes** from their statements and identified the key challenges faced by both sides. The interview questions were carefully designed to place participants within **dynamic scenarios**, allowing their responses to reflect the specific difficulties that arise in such contexts. The exact questions corresponding to these quoted responses can be found in **Appendix A** (for Senders) and **Appendix B** (for Receivers).

4.2.1 Theme 1: Dynamic Scenarios Characteristics

From what we have learned in our interviews, dynamic scenarios are complicated by **Time Pressure**, **Environmental Constraints**, **System Instability**, and **High Cognitive Loads**, which hinder Senders from submitting feedback. Time pressure causes Senders to postpone feedback, while environmental constraints, such as network problems, increase submission difficulty. System instability reduces trust, and a high cognitive load makes tasks take precedence over feedback, requiring the design of fast and non-intrusive systems.

Sub-theme 1.1: Time Pressure

> S2: "In such **high-stress**, **time-sensitive** moments, I usually **postpone** feedback or ignore it."
>
> S4: "I won't give feedback right then but if it's frequently happening, I will email them **afterward**."
>
> R7: "Users may **forget** details or provide incomplete reports due to time pressure and urgent tasks."

Time pressure in dynamic scenarios significantly hinders feedback submission. Senders prioritize primary activities over urgent tasks (such as navigation or medical emergencies), resulting in **delayed** or **ignored feedback** and a reduction in quality. Developers recognize that this not only affects the timeliness of feedback, but may also result in incomplete content, making it more difficult to identify problems. Designs need to allow for **delayed feedback** to ensure that Senders provide input when it does not affect the task. By simplifying the feedback process and providing immediate options, the system can reduce the negative impact of time pressure. Developers also recommend proactively prompting for feedback after the task is completed to capture insights that may get forgotten.

Sub-theme 1.2: Environmental Constraints

> S27: "If I'm in a place with a **bad signal**, I can't send feedback right away."
>
> R6: "Dynamic scenarios like **GPS errors** require systems to handle feedback without user effort."
>
> R8: "Dynamic scenarios like **driving** pose challenges as real-time user feedback is limited. Automated logs and system alerts are used instead."

Environmental constraints are also a significant barrier to feedback collection in dynamic scenarios. Unstable network signals, physical mobility restrictions, or other external

factors often prevent Senders from submitting feedback instantly. For example, in remote areas or environments with **weak signals**, Senders may not be able to send detailed reports, or even simple notifications. In **high-mobility scenarios** (such as walking or driving), it may be impractical or unsafe to stop to provide feedback, affecting real-time and the relevance of the feedback. Developers recommend developing systems that support **offline storage** and **automatic logging** to ensure that data is still captured when external conditions are limited. By allowing Senders to automatically upload feedback after the connection is restored or recording key events through background processes, the system can improve the robustness and completeness of data collection. These strategies ensure that feedback is not lost due to environmental constraints and enhance the system's adaptability.

Sub-theme 1.3: System Instability

> S8: "If the system **crashes**, I don't bother giving feedback because it feels pointless."
>
> R5: "End-user feedback is sometimes lost due to **system glitches**."

System instability undermines Sender trust and willingness to provide feedback in dynamic scenarios. **Crashes** or **technical failures** make Senders worry that feedback will not be processed, especially in emergencies. For example, application failures may lead to interruptions in feedback submission, which increases Sender frustration and reduces the likelihood of future feedback. Developers emphasize the importance of **system stability** and **reliability**, and recommend implementing strong error handling mechanisms and clear feedback receipt confirmations (such as "Your feedback has been received"). By separating feedback collection functions from the main application functions, main application failures can be prevented from affecting the submission of feedback. Stable system design is key to maintaining Sender engagement in dynamic scenarios.

Sub-theme 1.4: High Cognitive Loads

> S10: "Lack of time, **high mental load**, and not wanting extra distraction..."
>
> S25: "When I'm focused on a **critical task**, feedback is the last thing on my mind."
>
> R7: "Users may forget details or provide incomplete reports due to time pressure and **urgent tasks**."
>
> R9: "Feedback collection should not add to the user's cognitive load; it needs to be **seamless**."

High cognitive loads significantly affect feedback submission in dynamic scenarios. Senders perceive feedback as a **distraction** during complex tasks, such as driving or surgery, and prioritize primary activities. For example, doctors find it difficult to fill out feedback forms during emergency surgery, which leads to delayed or forgotten feedback and reduced data quality. Developers recommend **non-intrusive methods**, such as submission options after task completion or automated data capture, to reduce cognitive and operational burden. By embedding feedback into the workflow, the system ensures that the feedback process is aligned with Sender priorities. Developers also recommend using context-aware prompts to **trigger feedback** requests based on the Sender's task status to maximize the amount of information collected. The challenges of high cognitive load need to be addressed by designing feedback mechanisms that fit the Sender's workflow, requiring minimal effort or attention to finish giving feedback. This includes approaches such as automated data capture, post-task feedback prompts, or context-aware feedback

options that do not disrupt Senders' main task during their critical activities.

4.2.2 Theme 2: Feedback Collection in Dynamic Scenarios

In dynamic scenarios, feedback collection needs to be embedded in the Sender workflow to minimize interference, and strategies must be seamlessly integrated. Common feedback topics include usability issues and feature requests. Multimodal formats such as voice feedback meet the feedback needs of different Senders. Multiple channels for feedback, such as in-app forms, can improve convenience, and diverse inputs need to be processed efficiently.

Sub-theme 2.1: Feedback Collection Strategies

> S22: "In time limitation and stress I prefer One-tap/voice input within the application"
>
> R2: "For collecting real-time data of users I suggest Automated data logging: Instead of relying on manual input, **logs automatically** capture system behavior during critical events."
>
> R3: "Using **predefined** short messages or topics for issues, or simplified reporting **buttons** which can help in collecting structured feedback and better categorizing and routing can be helpful for Senders when they are in dynamic scenarios."

In dynamic scenarios, both **explicit** and **implicit** feedback collection strategies are adopted to obtain Sender input effectively. Explicit strategies include **direct Sender input**, such as clicking a button or filling out a short form, while implicit strategies capture data through **automatic logging** or **behavioral analysis**. For example, a navigation app can automatically record path deviations or error message interactions, providing developers with the necessary insights without requiring additional Sender action. The combination of the two improves the **quality** and **timeliness** of feedback and adapts to fast-paced environments. Simplified feedback methods not only increase the amount of data but also ensure contextual relevance. By optimizing these strategies, developers can enhance the efficiency of feedback management, support rapid problem identification and resolution, and improve the overall user experience.

Sub-theme 2.2: Typical Feedback Topics

> R1: "In unstable situations of environment such as unstable network connection, the feedback mainly includes **system errors** (bugs) and **feature requests**. Senders usually state, 'This part of the system is not working.'"
>
> R4: "While using autonomous systems for piloting a car as an on-the-go and unstable situation, the most common types of feedback are **system bugs** related to autonomous driving features. Issues such as emergency function failures, braking problems, and unexpected system behaviors are commonly reported."
>
> R10: "**Bug reports**, **performance complaints**, **desires for other features**, **usability issues** and **comparison** with similar products."
>
> S10: "I usually report **bugs** or **inappropriate content**. I provide feedback to help improve the user experience."
>
> S16: "Mostly, my feedback is about making the app or website **easier to use**. Since people from different backgrounds use the application, I believe it's much better if it is user-friendly and simple."

In dynamic software scenarios, user feedback focuses on several key areas that directly

affect **system performance** and **user satisfaction**. First, Senders frequently report system **errors** or **bugs**, especially in safety-critical applications such as autonomous driving. Problems such as emergency functional failures, braking issues, or unexpected system behaviors can lead to severe consequences, underscoring the importance of prioritizing safety. Second, Senders often make **feature requests** or **suggestions for improvements** to existing features, reflecting their expectations for system functional enhancements to improve practicality. Third, **usability issues** are an important concern, and Senders emphasize that the interface must be intuitive and easy to use, especially to provide a simple and friendly experience for Sender groups from different backgrounds. Fourth, **performance-related complaints**, such as slow response time or system instability, are prevalent, indicating that optimizing system efficiency is critical. In addition, Senders point out system deficiencies by **comparing** with similar products, providing valuable insights for competitive positioning and improvement. These feedback themes highlight the importance of balancing both functional and non-functional requirements in dynamic environments to ensure system reliability and user satisfaction.

Sub-theme 2.3: Multimodal Feedback Formats

> R4: "Feedback is mainly received in text format via **logs** and **structured reports**. In some cases, screenshots and images are uploaded when Senders report an issue, when they go to a workshop or service point and a technical person in that point will add screenshots to the text report or logs."
>
> R9: "Feedback is received in various formats, including **text**, **audio**, **screenshots**, and **logs**. The specific format depends on the context of the feedback. For instance, sensor data such as steering angles, acceleration, and jerk are used to analyze driver reactions... using eye-tracking cameras and accelerometers to gather more detailed user behavior data."
>
> R10: "We invite the users to the truck, we make **videos** collecting the feedback/ From other areas like forums we just pick the **text**. Sometimes we pick YouTube reviews as well."
>
> S1:"**Text** + **pictures**, also rating is easy for me to give."
>
> S16: "I prefer to submit my report in writing by sending a **text** or **email**, along with screenshots and photos, to document my feedback properly."
>
> S28: "In-app **rating** or emojis, send **voice** or **call** with support center."

In dynamic software scenarios, the multimodal **formats** of user feedback significantly enhance the depth and breadth of **data collection** and provide diverse perspectives for system improvement. **Textual feedback**, such as logs, structured reports, and emails, is the most common form, supporting detailed problem description and analysis. **Images** and **screenshots** are particularly suitable for visual problem descriptions by intuitively displaying interface problems or error scenarios, reducing barriers to developer understanding. **Audio** feedback, including voice messages and support center calls, provides a convenient way of expression in scenarios where Senders' hands are busy or moving, and is suitable for quick feedback. **Video** feedback is particularly suitable for diagnosing complex dynamic problems such as autonomous driving by recording event sequences. Ratings and emoticons are quick emotional feedback tools that allow Senders to express satisfaction or problem severity when time is tight.

In safety-critical applications, **sensor data** (such as steering angle, acceleration, and eye tracking) provides objective behavior and system performance indicators to supplement the shortcomings of subjective feedback. The integration of these multimodal formats

requires UFRM to design flexible processing mechanisms to adapt to inputs in different formats, ensure the comprehensiveness and accuracy of feedback, and support system optimization and user experience improvement in dynamic environments.

Sub-theme 2.4: Variety of Feedback Channels

> R3: "Feedback is collected mainly through **support tickets, emails, live chat**, and **internal reporting systems**. Some feedback also comes from **app store reviews** and website surveys."
>
> R8: "Users provide feedback via various channels, including **in-app reporting**, **customer service calls**, and **dealership reports**."
>
> R10: **"Live interviews, customer service, forums**."
>
> S7: "I usually prefer to give feedback through direct customer support channels like **live chat** or **email**, especially when I need a quick resolution."
>
> S15: "I prefer text, **In-app forms**, **chatbots**."

In dynamic software scenarios, the diversity of **feedback channels** ensures that Senders can submit feedback conveniently in different situations. **Support tickets** and **emails** are suitable for addressing complex issues, as they provide detailed records for easy tracking and analysis. **Live chat** and **customer service phone** provide instant support in emergency or mobile scenarios to meet the need for rapid response. **Internal reporting systems** and **dealer reports** capture feedback in professional scenarios, such as service point data in the automotive industry. **App store reviews** and **website surveys** reflect public opinion and provide developers with a broad market perspective. **In-app reports and forms** are seamlessly embedded in the user experience, reducing operational burden and increasing engagement. On-site interviews and forums promote in-depth interactions and reveal detailed insights. **Chatbots** efficiently handle simple queries and save resources. The integration of these channels enables UFRM to capture the user experience fully and ensure the timeliness and diversity of feedback.

4.2.3 Theme 3: Feedback Motivation in Dynamic Scenarios

User feedback motivation is driven by personal interests, altruism, emotional state, and trust. Personal interests and system improvements motivate feedback. Feedback is more likely to occur when emotions are high, simplifying interactions reduces effort, and privacy and data trust are critical, so timing and trust mechanisms need to be optimized.

Sub-theme 3.1: Reasons for Giving Feedback

> S3: "If my feedback can help **fix this crash** quickly, I'll definitely report it."
>
> S14: "Just to prevent the same scenarios happening for **other people**, especially in urgent situations."
>
> S20: "Getting help to **resolve my login issue quickly** and maybe receiving some small **voucher or discount**."
>
> S25: "I shared feedback because during **emergencies**, even small fixes can make a big difference."
>
> R2: "When the application **fails or crashes during peak hours**, users will report the problem because it needs to be **reliable at these times**."

Senders provide feedback in dynamic scenarios for reasons of self-interest, altruism, and external incentives. **Self-interest** drives Senders to solve their problems, while **altruism** motivates them to improve the system for the benefit of others. For example, a Sender may submit feedback because they want to fix a navigation bug, while expecting to improve system performance. **External incentives** such as discounts further enhance participa-

tion. Developers need to increase motivation by highlighting the outcomes of feedback and providing clear incentives. For example, showing how user feedback leads to updates can increase willingness to participate.

Sub-theme 3.2: Good Time and Mood to Submit Feedback

> S5: "If I am **in the mode** after busy, I will provide my feedback."
>
> S14: "When I'm stuck in traffic or waiting, I sometimes open the app and **report issues I faced earlier**."
>
> S18: "I give feedback after a **frustrating issue**, when I still feel annoyed if it's easier to describe it clearly."
>
> S22: "I always give feedback immediately after a **good experience** of using the app. Normal experience is not worth for taking time. Of course I give a poor note for a **bad experience**."
>
> S24: "e.g. amazon sends mail, and request for feedback, then if I have enough time, i provide an average feedback. Whenever it is super **exciting**, and is more than my expectation, I try to share a perfect positive feedback."
>
> S26: "When I'm rushed or stressed, I don't make the effort to provide feedback."

The **timing** and **mood** of feedback submission are important in dynamic scenarios. In some cases, feedback senders give feedback when they have free time or after completing a task, because they can think clearly. But in urgent or stressful situations, some senders submit feedback during the task, especially if the issue is serious and affects what they are doing. Strong emotions, such as excitement, frustration, or surprise, often lead to feedback. These emotions are common in dynamic settings, where things change quickly and users must react fast.

Feedback systems should support different situations. When users are busy or under stress, the system should allow simple and fast feedback. When users have more time, the system can ask for more detailed input. This helps collect feedback that is both timely and useful.

Sub-theme 3.3: Feedback Interaction Effort Minimization

> S1: "**Not disruptive** for the main task and **Ease of Access** are the most important for me."
>
> S4: "Make it as **easy** as possible, reflect directly even if it's only a confirmation."
>
> S8: "something **simple**, so I just need to click and then could finish giving the feedback."
>
> S9: "The **easier** and **shorter** the form is, the better."
>
> S13: "Under dynamic scenarios, **auto tracking** is better."

Minimizing the **interaction cost** of feedback is key to increasing engagement in dynamic scenarios. Senders prefer **simple**, **fast methods**, such as one-click options or automatic tracking, to reduce cognitive and operational burdens. For example, an app can provide "satisfied" or "unsatisfied" rating buttons that allow Senders to complete feedback in a few seconds. Developers need to design an **easy**, **non-intrusive** process to ensure that it does not interfere with the main task. By reducing the interaction cost, the system significantly improves the feedback submission rate and quality. Developers also recommend using **context-aware** technology to adjust feedback requests based on Sender status, ensuring the process is seamless and efficient.

Sub-theme 3.4: Trust and Privacy in Feedback Submission

> S7: "It is needed to follow, Automatic Tracking, Timeliness, Sender **Privacy** Consideration."
>
> S16: "Maintaining **privacy** without requiring Senders to reveal their identity can be important."
>
> S29: "**privacy** without requiring is important, the company should ensure me that they use my data with privacy and my control. I should know where my data is used. Where is it stored? maybe if they keep it in the car that I am working with, that's fine, But if they take my data to somewhere else and I don't have any Control over that, No, I don't like."
>
> R2: "Users are notified via agreements regarding data **usage policies**. Privacy protection: External tools cannot be used to process Sender data."

Trust and **privacy** issues significantly affect feedback willingness in dynamic scenarios. Senders are concerned about **data security** and **control**, and may avoid submitting feedback due to privacy concerns. For example, Senders in medical applications may be sensitive about data sharing. Developers recommend implementing clear **data use agreements** and **anonymity** options to enhance trust. By clearly communicating how data is stored and used, the system can alleviate concerns and encourage participation. Developers also need to ensure that data protection measures are in place, such as encrypted storage, to build Sender confidence. By addressing trust and privacy issues, organizations create a safer feedback environment and promote higher participation.

4.2.4 Theme 4: Feedback Internal Processing and Workflow in Dynamic Scenarios

Fast-paced settings like autonomous driving demand sharp feedback handling. This theme discussing feedback processing manners like sorting, prioritizing, and routing Sender input to the right teams quickly. Smart workflows cut delays, keeping systems humming. UFRM taps these strategies to make feedback processing smooth and reliable.

Sub-theme 4.1: Feedback Processing Strategies

> R2: "Feedback analysis is conducted using **Python-based** customized tools, tailored to different business requirements."
>
> R4: "**JIRA** and **internal support ticketing systems** are used to track feedback, categorize issues, and assign them to appropriate Receivers."
>
> R5: "They use both **automated** and **manual tools** to analyze feedback. Some tools perform root cause analysis, like fish bones and automated testing, while manual validation by professionals ensures accuracy."
>
> R7: "They use **internal monitoring systems** and **Confluence** for documentation. The Receiver **manually** logs and tracks issues, ensuring knowledge is shared effectively across teams."

Feedback processing in dynamic scenarios requires a balance between **automated tools** and **manual supervision** to handle large amounts of feedback efficiently. **Automated** tools such as **Python scripts** or **JIRA** help categorize and assign issues, while **manual verification** ensures the accuracy of complex issues. For example, a company might use an automated system to screen feedback, and then have experts review high-priority issues. This combination optimizes speed and depth to adapt to rapidly changing

business needs. By integrating automated and manual processes, organizations can respond quickly to Sender needs while ensuring processing quality and supporting system improvements.

Sub-theme 4.2: Feedback Categorization

> R2: "Feedback processing varies based on the **attribute type**: Stakeholder reactions are prioritized over Sender reactions, as direct Senders may not always be the primary decision-makers."
>
> R5: "Feedback is categorized based on **severity**, **frequency**, and **business impact**. They have a structured workflow to ensure critical issues get addressed first."
>
> R8: "Feedback is categorized and analyzed using structured processes. **Safety-related** issues are given top priority before other types of feedback."
>
> R10: "We categorize via severity of the tickets, in the severity we evaluate if this can **cause danger** to our customer and to the brand."

In dynamic scenarios, **feedback classification** mechanisms organize complex inputs through multi-dimensional criteria, laying a structured foundation for system improvement. We found that the classifications are significantly different from company to company based on their products and services. They are usually decided based on attribute type to distinguish between stakeholders and general user feedback. This ensures that the unique perspectives of different roles are captured, such as decision makers' concerns about strategic issues. We found some classification methods, for example, textbfSeverity classification assesses the potential impact of feedback issues, such as the direct consequences of functional failures on Senders. **Frequency** classification identifies recurring problems, revealing common defects or Sender pain points in the system. **Business impact** classification measures the significance of feedback to organizational goals or brand reputation, helping developers focus on key areas. **Safety-relevance classification** specifically targets high-risk issues in scenarios such as autonomous driving, including brake failures, to ensure that safety issues are accurately identified. UFRM needs to design a flexible classification framework that can adapt to the diversity of feedback in dynamic scenarios, ensuring efficient and accurate data organization and supporting system optimization.

Sub-theme 4.3: Feedback Prioritization

- Prioritization based on **severity** and **potential danger**: R1, R3, R4, R5, R6, R8, and R10 all mention severity as a prioritization criterion, and R4, R8, and R10 particularly emphasize potential dangers (such as user safety and passenger safety).
- Prioritization based on **business impact**: R2, R3, R4, R6, and R8 mention business impact, and R2 particularly emphasizes the principle of business priority, which is suitable for business-driven scenarios.
- Prioritization based on **frequency**: R3, R5, R6, and R8 mention frequency, focus on the prevalence of problems, and affect prioritization.
- Prioritization based on **safety and risk**: R4, R5, R8, and R10 mention safety and risk, and R5 particularly adopts the FMEA method, focusing on risk rate and detectability.
- Prioritization based on **user impact**: R1 and R3 mention user impact, focusing on the impact of feedback on user experience or user groups.
- Prioritization based on **urgency**: R4 mentions urgency, which is suitable for scenarios that require rapid response.
- Prioritization based on **functional relevance and system attributes**: R2 mentions functional relevance and key system attributes, which is applicable to technology-driven systems.

In dynamic software scenarios, the **feedback prioritization** mechanism optimizes the order of problem handling based on multiple criteria to ensure efficient resource allocation. **Severity and potential danger** assess the impact of the problem, such as brake failure in autonomous driving, highlighting user safety issues that are most concerning to each developer. **Business impact** measures the role of feedback on organizational goals and brand reputation, and also has an important influence on the prioritization. **Frequency** identifies repeated issues, which can reveal the prevalence of system defects and promote batch repairs. **Safety and risk** assessment, combined with failure mode and effect analysis, prevents dangers in high-risk scenarios, such as functional failures, and is also an important factor in ensuring product quality. **User impact** focuses on the role of feedback in enhancing the experience or satisfaction of a user group, thereby optimizing overall satisfaction. **Urgency** responds to scenarios that require a rapid response, such as real-time functional issues, emphasizing the need for a swift response. Functional relevance focuses on core system attributes to ensure that technical problems are resolved, making it suitable for technology-driven systems. Each criterion plays a unique role in different scenarios and together supports a structured workflow. UFRM needs to design a flexible priority framework to integrate these criteria, ensuring a rapid response to key issues in dynamic scenarios.

Sub-theme 4.4: Feedback Process Timing and Routing

> R4: "There is a structured approach where feedback related to different functions is assigned to **specific teams**, showing a systematic routing method to ensure that feedback reaches the relevant responsible team in the **shortest time**."
>
> R6: "Feedback is routed based on its nature, indicating that the **hierarchical routing** mechanism can quickly match the appropriate processing level according to the type of problem and **reduce processing delays**."
>
> R8: "Feedback is routed to different teams based on component responsibility, highlighting the **precise routing** based on the division of responsibilities to ensure that each problem has a clear responsible team."

In dynamic scenarios, **timely routing** and **processing** reduce delays and ensure rapid resolution. Feedback is assigned to specific teams based on the type of issue or component. For example, security-related issues may be routed directly to a team of experts, while functional issues are assigned to a product team. **Structured workflows** and **clear responsibilities** increase efficiency and reduce bottlenecks. Developers emphasized that a systematic routing approach is the core of efficient processing in dynamic scenarios. For example, a company may use an automated tool for initial assignments, followed by manual confirmation of complex issues. By optimizing timing and routing, organizations can maintain high response rates and meet the needs of dynamic environments.

4.2.5 Theme 5: Feedback Quality Limitations in Dynamic Scenarios

In various dynamic scenarios such as emergency response or driving, user feedback can be vague, delayed, or irrelevant due to haste, slowing down the pace of system improvement. Structured forms, instant collection, and filtering tools can significantly improve data quality. Ensuring that feedback is of high quality, clear, and practical is crucial for driving rapid system upgrades and enhancing the user experience.

Sub-theme 5.1: Vague or Incomplete Feedback

> R1: "Challenges include limited feedback volume, inconsistent quality, **vague or unclear** feedback, and partial capture of the user experience. When feedback is sent in the middle of a shift, users only write things like 'it crashed' or 'didn't work' without any clear context."
>
> R2: "We receive many short reports during peak hours, but they miss the sequence of actions that caused the failure."
>
> R5: "Processing feedback is complex due to missing information, **ambiguous reports**, and the difficulty of identifying root causes in large systems."
>
> S9: "I was in a rush, so I just typed 'failed to load' and skipped the details."
>
> S13: "I can't focus on explaining the situation and writing it down."
>
> S22: "It was during a delivery run, and I didn't have time to explain everything. I just tapped submit quickly."

In dynamic scenarios, **vague or incomplete feedback** often happens because feedback senders are busy or in a hurry. When senders give feedback during tasks like delivery, work shifts, or commuting, they may only write short messages like "crashed" or "didn't work." These reports often miss important details, such as what they were doing before the problem happened. **Vague feedback** is unclear, and **incomplete feedback** misses key steps or context. This makes it hard for developers to understand and fix the issue. Sometimes users skip the details because they don't have time or because the app doesn't guide them on what to write.

To improve this, feedback systems are better to offer simple forms with choices or short questions. This helps users give useful feedback even when they are busy. It also helps developers get better information without guessing what went wrong.

Sub-theme 5.2: Irrelevant, Redundant, or Conflicting Feedback

> R2: "**Dirty data and noise**: Processing raw feedback directly is time-consuming and reduces accuracy."
>
> R3: "Challenges include **irrelevant** or low-quality feedback, **conflicting feedback** (from different Senders)... Also they always face with spam or fake feedback."
>
> R5: "Challenges include feedback overload, duplicate reports, **irrelevant** or false reports, and conflicting information."
>
> R7: "One major challenge is that customers and users often have **conflicting needs**. Process owners might want stability, while users request changes."

In dynamic scenarios, irrelevant, redundant, or conflicting feedback reduces the applicability and consistency of feedback, thereby increasing the processing burden. **Irrelevant feedback** deviates from system issues, such as Senders submitting emotional catharsis rather than technical details. **Redundant feedback** manifests itself as repeated reports of the same problem, wasting resources. **Conflicting feedback** stems from differences in Sender needs, such as some Senders requiring a stable system while other Senders request new features, making it difficult for developers to make unified decisions. Spam will further interfere with analysis and affect data quality. UFRM needs to develop **filtering** mechanisms, such as keyword matching or sentiment analysis, to remove this feedback. Deduplication tools can identify duplicate reports, and conflicting requirements need to be prioritized through user group analysis. These mechanisms ensure that feedback con-

tent aligns with system improvement goals, thereby enhancing processing efficiency and decision accuracy.

Sub-theme 5.3: Unstructured Feedback

> R3: "Most feedback forms and tickets have a **freetext field** that can contain anything and should be analysed manually... they are working on forms that are just multiple choice so that the feedback can be automatically categorized."
>
> R4: "A lot of time goes into figuring out what users actually meant. Feedback can include anything, for example, actual bug reports to vague complaints about performance. so in time limitation, **structured feedback can help to process sooner**."
>
> R6: "Key obstacles in feedback processing include **lack of structure**, and difficulty in diagnosing issues."
>
> R7: "Users provide input mainly through informal discussions, but some feedback is also gathered through **structured** reporting systems."

In dynamic scenarios, **unstructured feedback** becomes another obstacle to effective processing. Unlike vague or conflicting feedback, which relates to content clarity or consistency, unstructured feedback refers to the lack of a defined format or input structure. This includes feedback written in open-ended text fields, messages sent via informal channels, or data scattered across multiple sources without alignment. Such feedback is difficult to process efficiently, especially in high-pressure environments where quick understanding is essential. Developers must read through varied and unpredictable formats, which increases manual effort and delays response time. In dynamic settings, this can slow down system improvements or real-time adjustments. To reduce these delays, UFRM encourages the use of structured forms and predefined input categories to help users provide clearer and more consistent input. This structure also supports automation tools, making it easier to extract useful information for decision-making.

Sub-theme 5.4: Delayed or Missed Feedback

> R1: "Users often **do not provide feedback** while driving due to safety concerns. Real-time feedback is difficult to collect in this scenario."
>
> R5: "Processing feedback is complex due to **missing information**, ambiguous reports, and the difficulty of identifying root causes in large systems."
>
> R9: "Feedback **is not submitted** during dynamic activities... Also they will be collected indirectly through the monitoring system and millions of sensors inside the car."
>
> S4: "I won't right then but if it's frequently happening, I will email them afterwards."
>
> S26: "When I'm rushed or stressed, I don't make the effort to provide feedback."

In dynamic scenarios, **delayed** or **missing feedback** submission stems from the Sender's task priorities in high-pressure environments. Senders often **postpone** feedback because they focus on their primary tasks, such as driving, responding to emergencies, or operating in time-critical environments, which might result in forgotten or untimely information. **Missing feedback** is common in safety-critical scenarios, where Senders completely abandon submission due to safety concerns or time constraints. For example, a driver may neglect to report a navigation error because they are focused on the road, or only submit a report after the problem recurs. Delayed feedback reduces real-time performance, and missing feedback leads to incomplete data, hindering developers from capturing key issues. UFRM needs to design an **instant** feedback mechanism, such as one-click report-

ing or voice input, to lower the submission threshold. Automatic sensor data collection can supplement user feedback that has not been submitted, ensuring data timeliness and comprehensiveness.

4.2.6 Theme 6: Feedback Follow-up and Response in Dynamic Scenarios

In a dynamic scenario, Senders want feedback to be **responded** to quickly to maintain their enthusiasm for participation. This theme examines how timely confirmation, actionable solutions, and feedback results can foster trust and promote ongoing collaboration. Personalized follow-up and regular updates further bring Senders closer to the system, building a deeper connection. These strategies ensure that the feedback loop provides lasting motivation for system improvement by building trust and a sense of value.

Sub-theme 6.1 Timely Acknowledgment

> S7: "I prefer to receive at least **an acknowledgment** that my feedback has been received."
>
> S10: "It's somewhat important — a **short confirmation** or update shows my feedback was noticed, but I don't expect a detailed response."
>
> S18: "I would definitely prefer my feedback being read and at least I would like to receive feedback (even: thanks for your feedback) makes me feel valuable."

Timely acknowledgment is critical to maintaining Sender engagement in dynamic scenarios. Even **brief notifications**, such as "Thank you for your feedback," can build trust and reduce alienation. For example, companies can use **automated systems** to send confirmation messages immediately after receiving feedback, ensuring that Senders know that their input has been received. Timely confirmation increases user confidence and encourages future feedback submissions. By ensuring that every piece of feedback is responded to promptly, the system can establish a positive feedback loop that enhances user satisfaction and increases their willingness to participate.

Sub-theme 6.2 Informative and Actionable Responses

> S1: "I hope I can get a **feasible solution**."
>
> S16: "It is very important that they respond in time and provide a **convincing explanation**."
>
> S24: "A more **meaningful response** is to respond to your feedback like a human, which encourages you to share more feedback, more accurately and precisely."

Senders seek practical solutions and clearly explained **responses** in dynamic scenarios. For example, developers can provide personalized responses, such as "Your report has been recorded and we will fix it in the next version", to **enhance trust**. **Informative and actionable responses** increase satisfaction and encourage more accurate feedback in the future. The system should focus on providing specific, actionable responses to meet user expectations. Developers can adjust the response content based on the feedback type to ensure that it is relevant to Sender needs, provides high-quality responses, enhances user trust, and promotes continuous participation and system improvement.

Sub-theme 6.3 Visibility of Feedback Impact

> S11: "I would give feedback if I can feel it can have an **impact**."
>
> S20: "We should be able to provide the feedback very fast and know/feel that they are going to do some **acts** based on my feedback."
>
> S23: "It is important for me, because I can see that my comments have been valued by that company."
>
> S28: "Also, if they can show my feedback's **impact** on the system, maybe it is a motivation."

Showing the **impact** of feedback is effective in motivating Senders in dynamic scenarios. For example, publishing change logs or notifications to show how user feedback leads to system improvements enhances the sense of value. For example, apps can show Senders how suggestions are adopted to encourage continuous feedback. Developers can highlight feedback results through regular communication, such as feature updates or bug fixes, to reinforce the importance of Sender contributions. By highlighting results, the system builds long-term engagement relationships.

Sub-theme 6.4 Continuous Engagement

> S3: "I expect a **follow up**."
>
> S11: "I expect a **follow-up** or solution and acknowledgment."
>
> S13: "Receiving **follow-up** helps me solve the problem, or a response that shows they care about my feedback is good."

In dynamic scenarios, **continuous engagement** ensures that Sender issues are resolved and enhances Sender enthusiasm for feedback through follow-up and long-term interaction. For example, both **S3** and **S11** said that Senders expect further follow-up after initial confirmation, such as updating the status of the problem through email or in-app messages. This **continuous communication** can make Senders feel that the problem is taken seriously. **S13** pointed out that follow-up not only helps to solve problems, but also enhances Senders' sense of belonging by showing that the company attaches importance to feedback, such as providing instructions for processing progress or solutions, which can effectively improve user satisfaction. **S30** expressed the hope that real people would follow up, indicating that the sense of care and urgency conveyed by personalized responses is particularly important in dynamic environments, and can motivate Senders to stay involved better than automated notifications. In dynamic scenarios, the feedback system should maintain contact with Senders through regular status updates, manual follow-ups, and problem resolution notifications to ensure that the issue is properly addressed, which can enhance trust and motivate long-term participation.

4.3 Insights From Thematic Analysis Results

This section presents the analytical outcome of **Step 5: Conceptualization**, where we interpreted the six core themes and their sub-themes to extract deeper insights. We identified feedback senders preferences and challenges (RQ 1.2), and receivers' preferences and challenges (RQ 1.3) in this section, based on the interview and thematic analysis results in the `Section 4.2`, and summarized the results in the `Section 5.2` when answering the RQs.

4.3.1 Identifying and Handling Feedback Barriers in Dynamic Situations

In many interviews, **feedback senders** described their **challenging situations** where they wanted to give feedback but could not do so due to their environment. These include being on the move, facing unstable systems, dealing with network issues, or being under time pressure, etc. Several senders reported **skipping feedback** entirely when the environment made it hard to act. In such cases, feedback is not ignored because feedback senders lack opinions, but because the situation prevents them from engaging. From our interviews, we realized that the feedback receivers tended to ignore this potential user feedback, and current feedback systems can fail in these dynamic contexts.

Insight:

- Feedback is often **skipped** when feedback senders face time pressure or system instability.
- Real-world conditions like **weak signal** or **app freezing** often prevent Sender input.
- Senders are more likely to skip feedback when it interrupts their task flow.
- Senders **may not remember** to report an issue after the situation has passed.
- Senders want the system to "know" when something goes wrong, even if they can't report it themselves.

Design implication:

- The system should **detect dynamic conditions** (e.g., GPS loss, network issues, Sender task load) in real time.
- Feedback mechanisms should only **activate** when context allows for it (e.g., not during a dangerous driving moment).
- Passive signals, such as system crashes or feature freezes, should trigger **automatic data collection**.
- The design must include mappings between scenario types and appropriate feedback mechanism configurations at **design-time**.

4.3.2 Making Feedback Sending Easy and Timely for Feedback Senders

Even if Senders are willing to provide feedback, the way the system asks for feedback and the effort required can be a **barrier**. Many interviewees mentioned that they **ignored** feedback prompts because they appeared at the **wrong time**, required **too much input**, or **disrupted their focus**. In high-pressure or fast-moving scenarios, Senders may feel that giving feedback is a **burden**, especially if they believe it won't lead to action. Some Senders prefer giving feedback immediately after a task, while others are open to doing it during the task if the interaction is instant. These findings highlight the importance of designing lightweight, flexible, and adaptive feedback collection mechanisms.

Insight:

- Complex or long forms **reduce motivation** to participate.
- Some Senders prefer giving feedback after the task; others are open to in-task prompts if the **effort is low**.

- Senders respond more when feedback input feels **meaningful** and **easy**.

Design implication:

- Feedback mechanisms should support **multiple input modes**, such as voice, one-tap, or passive triggers.
- The system should offer **flexible timing options** to collect user feedback (e.g., after task completion or during natural pauses).
- Collection methods should **minimize user effort** by using pre-filled context and simple templates.
- Mechanism activation should consider both Sender state and scenario type to **avoid creating disruption**.

4.3.3 Improving Feedback Clarity and Streamlining Internal Processing

From the **feedback receiver's** perspective, **processing feedback** efficiently is a major challenge, especially when it is ambiguous, poorly structured, or misdirected. Feedback Receivers often have difficulty understanding what feedback senders mean and which Receiver is responsible for handling feedback. At the same time, Senders may not know how to describe technical issues clearly, especially when they are under pressure or unfamiliar with the system's terminology.

Some recipients said it was a waste of time to screen **irrelevant** feedback. Others said even a well-intentioned report could be difficult to process if it was missing some important **metadata**, such as device type, version, or time. In turn, senders often had no idea what details to include. This highlights the need for feedback receivers to have a mechanism that not only facilitates the collection of feedback but also supports receivers' **internal** organization, categorization, and prioritization when processing data.

Insight:

- Feedback is often too **vague** or **incomplete** to be useful.
- Receivers spend time **manually** sorting or clarifying reports.
- Senders often don't know what information is needed or how to describe problems.
- Important feedback may be missed or delayed due to unclear ownership or **team routing**.

Design implication:

- Feedback is better to be enriched with automatic **metadata** (e.g., timestamp, scenario type, system status).
- Senders can be guided with lightweight templates or selectable categories to **structure** their input.
- **Internal workflows** should be clear and support feedback classification (e.g., bug, suggestion), prioritization, and assignment.
- UFRM needs to allow receivers to process and act on feedback efficiently and **automatically** without depending on manual triage.

4.3.4 Building Trust Through Meaningful and Timely Responses

A common frustration expressed by **feedback senders** is the **lack of feedback** after they submit an issue or suggestion. Many said they never receive a confirmation or follow-up, which leads them to believe their input is ignored. As a result, they are less likely to provide feedback in the future, even when encountering real problems. On the other hand, **feedback receivers** explained that even when problems were solved, there was often no structured way to inform Senders of the results. This communication barrier created a weak feedback loop that resulted in a loss of value for both parties. A strong response mechanism is essential for closing the loop, reinforcing Sender motivation, and building long-term trust in the system.

Insight:

- Senders often **do not receive** any acknowledgment after submitting feedback.
- The absence of response reduces Senders' motivation to **report again**.
- Receivers lack mechanisms to **manage responses** or **track follow-up**.
- **Closing the feedback loop** increases transparency and trust.

Design implication:

- The system should automatically **generate acknowledgments** or status updates when appropriate.
- Responses should be **contextual** and **meaningful** (e.g., "Bug fixed in version 2.3").
- UFRM should support **tracking** of user-facing responses, including delivery status.
- Building a feedback loop requires intentional communication back to the sender.

4.4 UFRM Design

4.4.1 Process Flow in UFRM

The **User Feedback Reference Model** (UFRM) is a **conceptual framework** developed to address the challenges of collecting and processing user feedback in dynamic software scenarios. It is systematically constructed based on a six-phase **thematic analysis** of forty interviews with feedback senders and receivers (see `Section 3.2`), resulting in **six** major themes and **twenty-four** sub-themes (see `Section 4.1`).

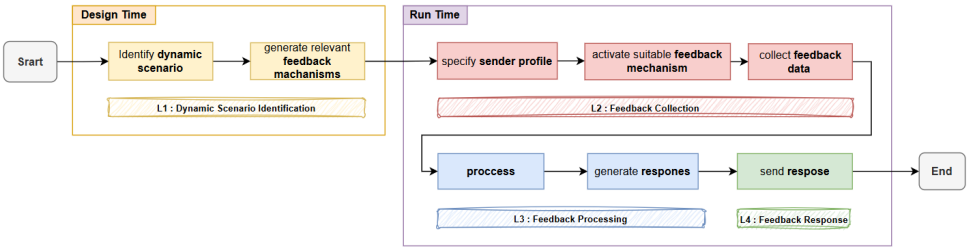

Figure 4.5: UFRM - Process Flow Diagram

Figure 4.5 illustrates the overall process flow of UFRM, organized into four sequential layers that together manage the full feedback lifecycle. The process starts at **design time** with **Layer 1**, where the system prepares necessary configurations for handling feedback under various dynamic scenarios. Once this setup is complete, the remaining steps occur during runtime.

At **runtime**, the flow continues with **Layer 2**, where the system identifies the active user and matches them with a relevant feedback mechanism. This allows the Sender to provide feedback in a way that suits both their factors and the current situation. The collected feedback is then passed to **Layer 3**, which processes and interprets the input, preparing it for decision-making. Finally, **Layer 4** delivers a response back to the Sender, ensuring the loop is completed.

Each layer builds on the previous one to ensure continuity and coordination across the system. The structure allows UFRM to operate flexibly while maintaining a clear, step-by-step progression from scenario detection to feedback resolution.

4.4.2 Entity Classes in UFRM

To make UFRM work in practice, we designed a clear and structured **UML class diagram** (see Figure 4.6). It shows the main **entity classes**, their related **attributes** and **lists**, how data flows through the system, and how different Senders and system components interact with each other during the feedback process.

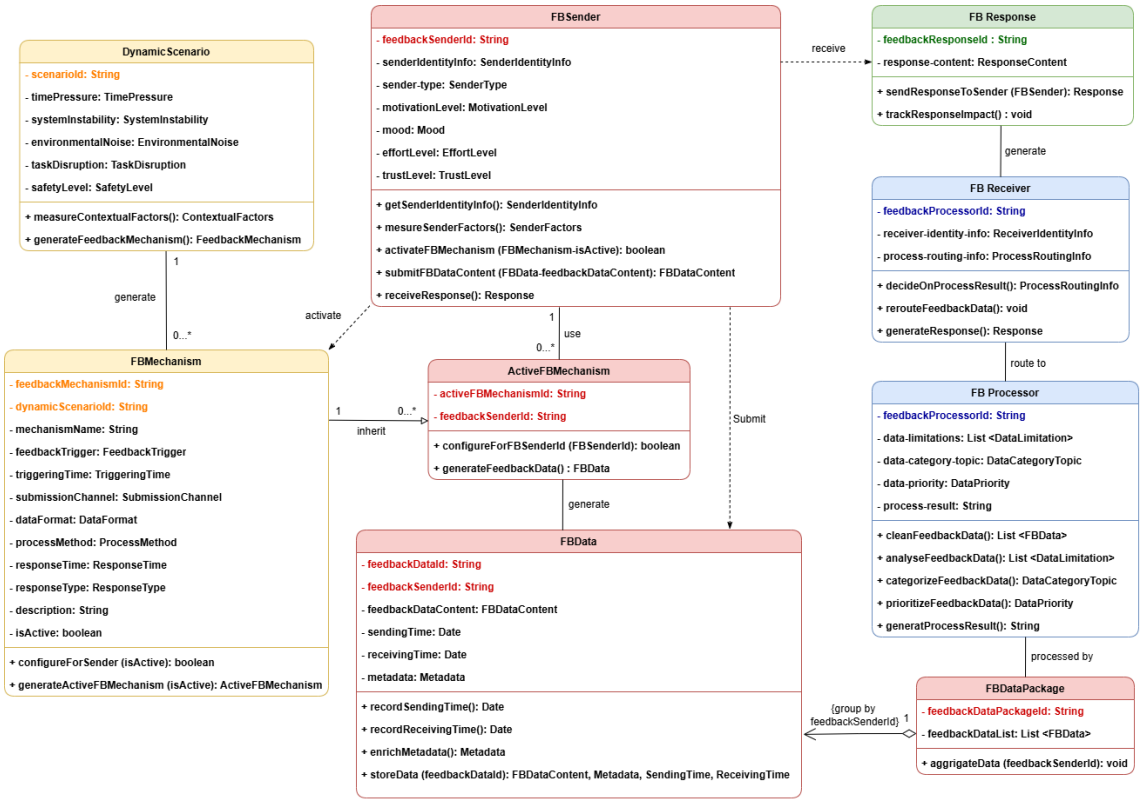

Figure 4.6: UFRM - UML Class Diagram

The model consists of **nine** key **Entity Classes**, each representing an essential component in the feedback loop. These classes contain attributes and methods, supported by multiple associated classes and lists that further define their values. We do not go into detail about each attribute in the main text here, but focus on the conceptual role of each entity and how it contributes to the layered architecture of the UFRM. The complete technical specification of each class, including attributes and associated classes and lists, is presented in `Appendix C` for reference. This structure ensures that both software designers and researchers can reuse or extend UFRM in future dynamic feedback systems.

From **scenario identification** to **updating feedback sender with response**, each entity class aligns with a specific layer in the model's process pipeline (L1 to L4, represented in different colors in the diagram), enabling structured feedback flow. For instance, `DynamicScenario` and `FBMechanism` initiate the process by defining contextual triggers and available configurations. These are followed by `FBSender`, `ActiveFBMechanism`, `FBData`, and `FBDataPackage`, which manage feedback collection. The processing is handled by `FBProcessor` and decision-making by `FBReceiver`, while the `FBResponse` class ensures the Sender receives a meaningful and timely reply.

4.4.3 Class Calls in UFRM Layers

To illustrate how UFRM operates at runtime, `Figure 4.7` shows the complete sequence of feedback processing through a concrete object instance. Each object corresponds to an **entity class** introduced in the UML class diagram (see `Figure 4.6`) and follows the **process flow** shown earlier in `Figure 4.5`.

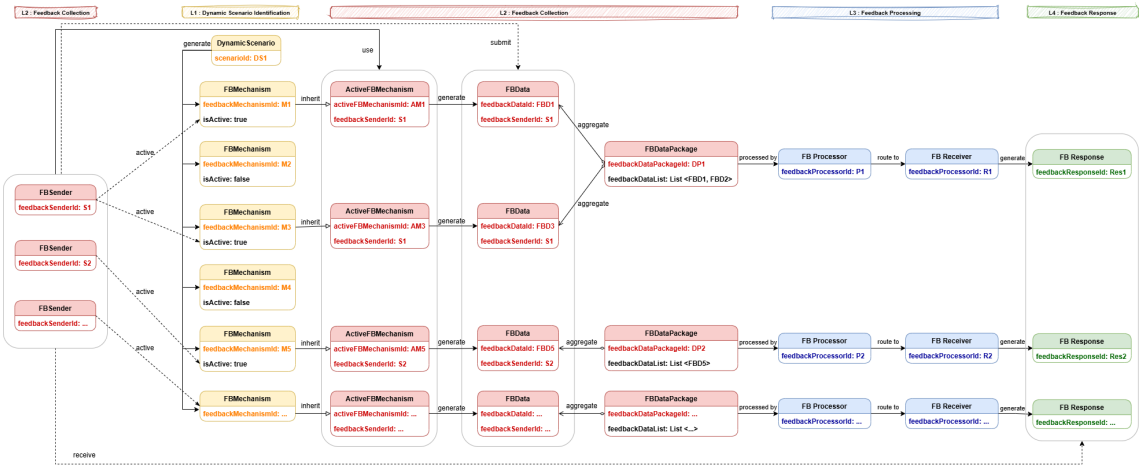

Figure 4.7: UFRM- Class Calls in Four Layers

Layer 1 – Dynamic Scenario Identification (Design-Time)

The system begins by identifying a dynamic scenario, such as `DS1`, using real-time contextual signals like high time pressure or system instability. This leads to the generation of multiple `FBMechanism` configurations (e.g., `M1-M5`) tailored to that scenario. These mechanisms specify submission settings such as format, timing, and trigger method. Only a subset (e.g., `M1, M3, M5`) is activated for later runtime use.

Layer 2 – Feedback Collection (Run-Time)

When a sender (e.g., `S1, S2`) enters the system, one or more of the active mechanisms are matched to their profile and instantiated as `ActiveFBMechanism` (e.g., `AM1, AM3, AM5`). The sender uses this to submit feedback, which is recorded as `FBData` objects (e.g., `FBD1, FBD2, FBD3, FBD5`). These entries include metadata such as submission time and device information. All feedback from the same sender is bundled into a `FBDataPackage` (e.g., `DP1, DP2`) to record submission history and simplify downstream processing.

Layer 3 – Feedback Processing

Each `FBDataPackage` is processed by a `FBProcessor` (e.g., `P1, P2`). The processor cleans the data, classifies each entry (e.g., bug, suggestion), and sets priorities (e.g., critical, minor). The result is routed to a `FBReceiver` (e.g., `R1, R2`), which makes a decision: resolve the issue, escalate it, reject it, or schedule it for future work.

Layer 4 – Feedback Response

Based on the decision, a `FBResponse` (e.g., `Res1, Res2`) is generated and sent back to the original sender. This response may confirm receipt, describe a resolution, or request clarification. It completes the loop and helps the system close feedback cycles promptly.

4.5 UFRM Validation

To assess the practical performance and adaptability of UFRM in dynamic settings, we conducted a structured validation process following the principles of `Design Science Research Methodology (DSRM)` [11]. DSRM recommends a **two-phase strategy** for validating design artifacts: **demonstration** and **evaluation**.

In the demonstration phase, we applied UFRM to **three real-world inspired scenarios** (Use Case 1-3 that are mentioned below) to show how the model works step by step in practice. Each use case was linked to the four layers of UFRM, showing how feedback is handled from detection to response. These cases reflect different types of dynamic environments, helping us test how well the model adapts to pressure, complexity, and Sender needs. **In the evaluation phase**, we reflected on whether the model achieves what it was designed to do. We checked if it supports fast, easy, and context-aware feedback handling under dynamic scenarios.

To help readers better understand how UFRM is used in real-world scenarios, we provide a detailed step-by-step walkthrough of `Use Case 1`. This includes a clear description of how values are selected, the classes used, and the methods called during the feedback management process. For the sake of brevity and clarity, `Use Cases 2` and `Use Cases 3` are presented in a more concise manner, focusing on the overall feedback handling flow rather than the internal mechanisms of each class.

4.5.1 Demonstrate Use Case 1: Lost Navigation During a Trip

1.1. Scenario Description

A passenger is using a navigation app while traveling through an unfamiliar city. Suddenly, the GPS signal is lost, and the map stops updating. The Sender becomes confused and stressed, unable to follow the route in real time. Because the passenger is mid-trip and

under time pressure, they may skip feedback. In this case, UFRM activates a voice-based in-app mechanism, allowing the Sender to quickly submit a voice message during the task without needing to stop or type. This reduces the effort and ensures timely issue capture.

1.2. Feedback sender interview excerpts

- S9: “When I realized the route was wrong, like it sent me into a dead end, I quickly noted it while still driving.”
- S16: “I was in a moving car, and the signal was bad. I couldn’t submit feedback until much later.”

1.3. UFRM Layer-Based Analysis

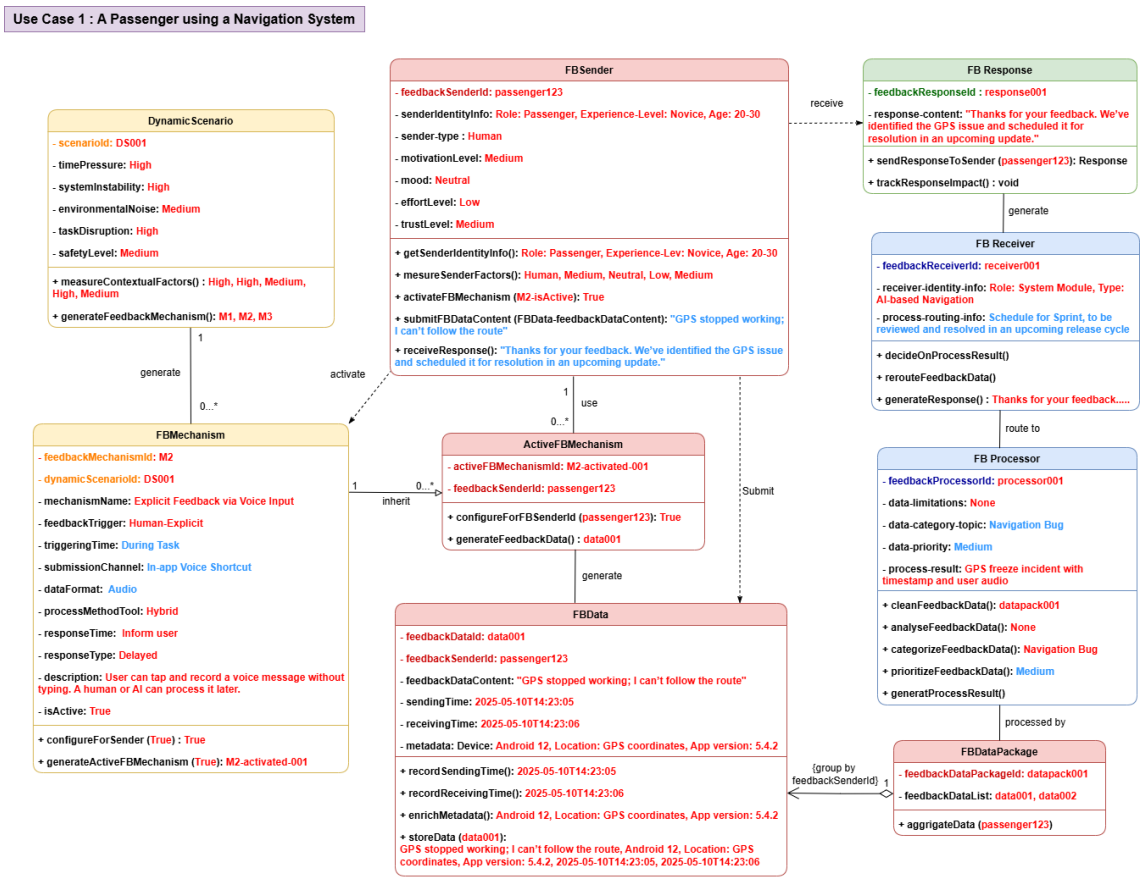

Figure 4.8: UML Class Diagram Demonstration for Use Case 1

L1 – Dynamic Scenario Identification

UFRM first identifies a dynamic scenario using the `DynamicScenario` class. Based on sensor logs and system status, it detects: `time-pressure = High`, `system-instability = High`, `task-disruption = High`, and `environmental-noise = Medium`. It then uses `generateFeedbackMechanism()` to create three predefined feedback mechanisms (`M1`, `M2`, and `M3`), each configured with different formats and submission methods. These are stored for later activation.

L2 – Feedback Collection

At run-time, the system analyzes the sender’s real-time context using the `FBSender`

class. The sender `passenger123` has attributes: `effort-level = Low`, `motivation = Medium`, and `trust-level = Medium`. Given this profile, `M2` (voice input) is activated via `activateFBMechanism()`. A new `ActiveFBMechanism` instance is created and linked to the Sender. The Sender submits voice input: "GPS stopped working; I can't follow the route." This is captured as an `FBData` object with metadata (time, location, app version), and combined with system logs into a single `FBDataPackage` using `aggregateData()`.

L3 – Feedback Processing

The `FBDataPackage` is processed by the `FBProcessor`, which executes: `cleanFeedbackData()`, `categorizeFeedbackData()`, and `prioritizeFeedbackData()`. The data is classified as a `Navigation Bug` and assigned a `Medium` priority. The final result is structured and passed to the receiver module for action.

L4 – Feedback Response

An `FBReceiver` (an AI module or support team) receives the processed result. It decides to schedule the issue for resolution in the next sprint and creates a response using `generateResponse()`. The resulting `FBResponse` message is: *"Thanks for your feedback. We've identified the GPS issue and scheduled it for resolution in an upcoming update."* This message is delivered to the sender, completing the feedback loop.

4.5.2 Demonstrate Use Case 2: Auto-Pilot Failure on Highway

2.1. Scenario Description

A patient is in a remote video consultation with a doctor. During the session, the doctor's voice cuts out, and the video freezes. The Sender is anxious and does not want to interrupt the session to report the issue. UFRM suggested presenting a one-tap survey pop-up to let the Sender report problems after the session. This mechanism reduces Sender effort and helps capture technical failures without disrupting sensitive healthcare interactions.

2.2. Feedback sender interview excerpts

- S10: "Lack of time, high mental load, and not wanting extra distraction…"
- S25: "When I'm focused on a critical task, feedback is the last thing on my mind."

2.3. UFRM Layer-Based Analysis

L1 – Dynamic Scenario Identification:

When the system detects a safety-critical event (e.g., auto-pilot lane change failure), it creates a `DynamicScenario` instance `DS002`, characterized by `High` time pressure and task load. Feedback mechanisms `M1` and `M2` are instantiated as `FBMechanism` objects.

L2 – Feedback Collection:

The sender `driver001`, modeled as an `FBSender`, is operating under stress but with high motivation. The system activates `FBMechanism M2`, represented as `ActiveFBMechanism M2-activated-002`. Voice input is used to submit the report: "Auto-pilot didn't react to vehicle ahead," recorded as `FBData`.

L3 – Feedback Processing:

Multiple `FBData` items are aggregated into an `FBDataPackage`, then processed by `FBProcessor`

`processor002`. The issue is classified as a `Safety Bug`, with `High` priority, and the system logs a detailed process result.

L4 – Feedback Response:

The `FBReceiver` (e.g., AI Safety Triage Module) routes the feedback to the compliance team and generates an `FBResponse` for the Sender: "Safety incident reported. Compliance team notified."

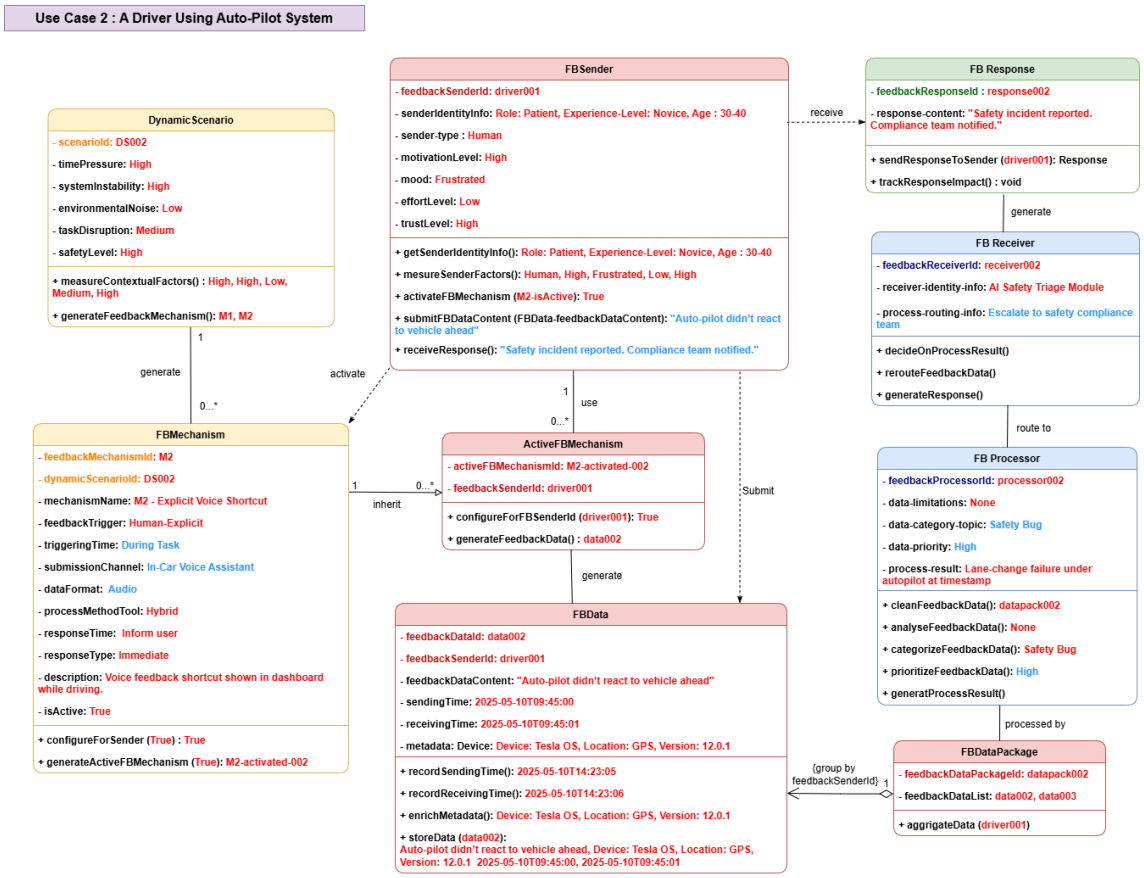

Figure 4.9: UML Class Diagram Demonstration for Use Case 2

4.5.3 Demonstrate Use Case 3: Interrupted Medical Video Call

3.1. Scenario Description:

A patient is in a remote video consultation with their doctor. During the session, the doctor's voice cuts out, and the video freezes. The Sender is frustrated and wants to report the issue, but doesn't want to interrupt the consultation. After the session, the patient forgot to report it. This scene combines emotional sensitivity, system instability, and environmental constraints. With UFRM, the system can detect technical failures in real time, log context, and generate a feedback record without requiring the Sender to act.

3.2. Feedback sender interview excerpts

- S8: "If the system crashes, I don't bother giving feedback because it feels pointless."
- S20: "If it freezes during a video meeting, I panic and ignore all popups."

3.3. UFRM Layer-Based Analysis

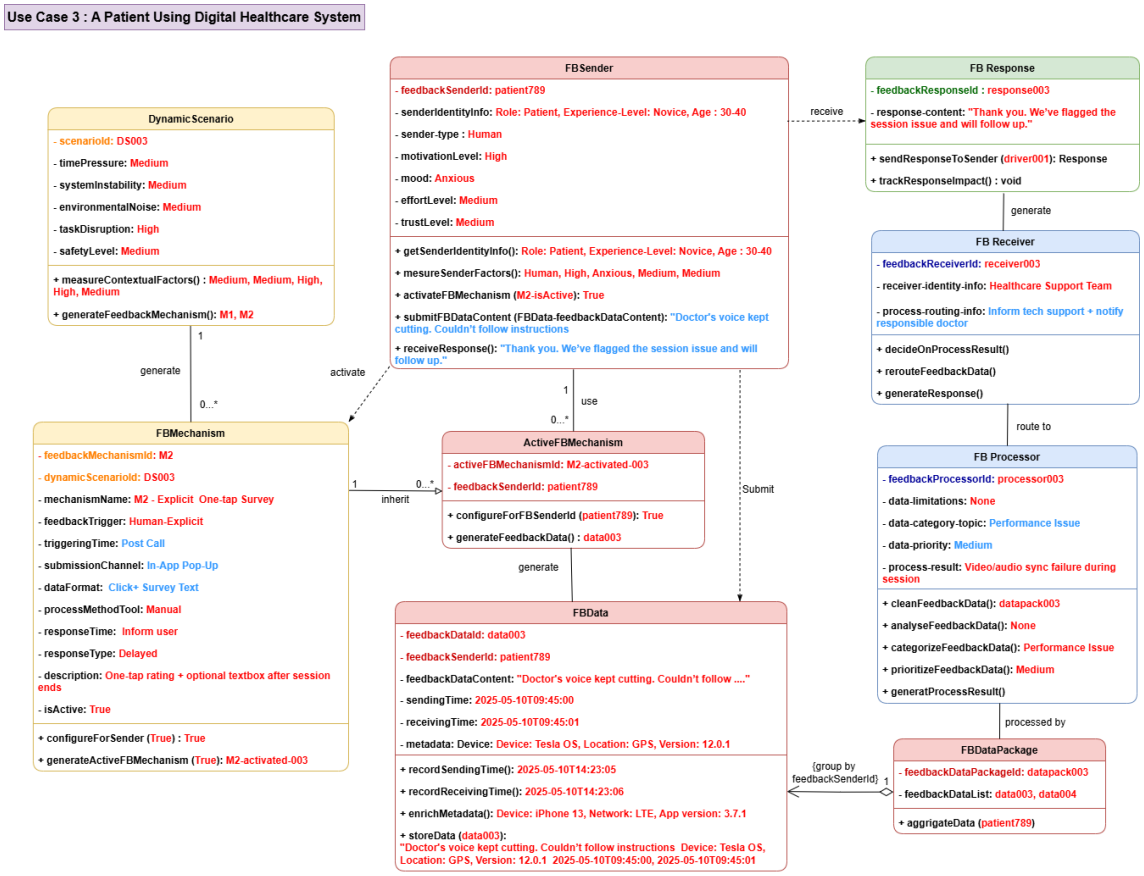

Figure 4.10: UML Class Diagram Demonstration for Use Case 3

L1 – Dynamic Scenario Identification:

The system detects that the Sender is in a high-disruption, emotionally sensitive context. It creates a `DynamicScenario DS003` with attributes like `Medium` system instability and `High` task disruption. Based on this, it generates `FBMechanism M2`, which is configured to activate post-call feedback.

L2 – Feedback Collection:

The Sender `patient789`, modeled as an `FBSender`, is given a `One-tap Survey` pop-up via the `ActiveFBMechanism M2-activated-003`. The Sender submits: "Doctor's voice kept cutting. Couldn't follow instructions," stored as `FBData`.

L3 – Feedback Processing:

The feedback is packed into a `FBDataPackage` and processed by `FBProcessor processor003`. The system classifies the input as a `Performance Issue`, assigns a `Medium` priority, and logs it accordingly.

L4 – Feedback Response:

The case is passed to the `FBReceiver` (e.g., Healthcare Support Team) responding via `FBResponse response003`: "Thank! We've flagged the session issue and will follow up."

4.5.4 Evaluate UFRM Design Goal

To evaluate whether UFRM fulfills its original design goals (see `Section 1.4.1`), we assessed its performance across the six stated objectives, using the three real-world-inspired use cases mentioned above in `Section 4.5.1 to 4.5.3`. This evaluation follows the approach outlined in our method (see `Section 3.3`), where model goals act as criteria for design assessment as a part of the DSRM [11].

The following paragraphs provide a goal-by-goal reflection, supported by examples from the use case demonstrations:

1. Context awareness

UFRM can react to different situations automatically. In the `Lost Navigation` case, the system noticed the loss of GPS and high time pressure. It then triggered a voice-based feedback method, so the Sender didn't have to stop or type. In the `Auto-Pilot Failure` case, the system recognized a safety-critical driving context and waited to collect feedback until it was safe to do so. This shows UFRM can adjust to the Sender's environment.

2. Low user effort

UFRM reduced the amount of work needed from Senders. For example, in the `Navigation` case, the Sender only needed to speak to give feedback. In the `Medical Video Call` case, a short one-tap survey was shown after the session. These simple methods helped Senders give feedback quickly, without adding stress or distraction during tasks.

3. Improved feedback quality

The model helps collect clear and useful feedback. In the `Auto-Pilot` case, the system added important event logs, such as when the lane change failed, so engineers could understand what happened. In the `Medical Video Call` case, the feedback included system information like network status and session ID. This extra context made the feedback more helpful to those who received it.

4. Efficient organizational processing

UFRM helps Receivers work with feedback more easily. In the `Navigation` case, the system marked the feedback as a route issue and noted that similar reports had come in. In the `Auto-Pilot` case, it saw the report was safety-related and sent it to the right Receiver. These examples show how the model helps classify, sort, and send feedback to the right place.

5. Closed feedback loop

UFRM makes sure Senders get a response. In the `Navigation` case, the system could later inform Senders that the route had been updated. In the `Medical Video Call` case, the Sender got a message saying the issue was being looked into. This kind of response builds trust and shows Senders that their feedback is taken seriously.

6. Support for both senders and receivers

UFRM is designed for both sides. Senders get simple ways to give feedback, like voice or one-tap input. Receivers get structured reports with helpful data and support for replying. For example, in the `Auto-Pilot` case, the system collected and sent technical info to the engineers, while also giving the Sender a simple way to confirm their report was received.

This balance makes the model useful for everyone.

5

Discussion

5.1 Key Differences Between UFRM and Traditional Feedback Management Systems

This section highlights how the User Feedback Reference Model (UFRM) differs from traditional feedback systems by directly addressing core weaknesses discussed in `Section 2.4`. These differences demonstrate UFRM's **contributions** to improving user feedback processes in dynamic software environments. The discussion is organized into four main areas: real-time handling, automation with contextual adaptation, sensitivity to dynamic scenarios, and cross-layer integration.

5.1.1 Real-Time Handling Across Feedback Management Stages

Traditional systems typically gather user feedback after events have occurred, either in batches or through delayed manual forms. This results in feedback that lacks context, making it harder for developers to understand what actually happened [46]. This challenge is especially critical in time-sensitive scenarios, such as autonomous driving or healthcare support, where details can be forgotten quickly or are no longer relevant. As mentioned in `Section 2.4.3`, this delay often results in missed feedback opportunities when Senders prioritize their task over providing feedback.

UFRM addresses this limitation by enabling real-time feedback across all stages, from problem detection to response, while the Sender is still engaged in their task. This approach preserves rich contextual information and shortens the delay between issue occurrence and system response. For example, during a live navigation error, Senders can submit voice feedback instantly, and the system logs contextual data such as GPS status and timestamp. This supports proactive system correction and user trust, which is not possible with traditional feedback loops [13].

5.1.2 Automatic and Context-Driven Decision Making

One major limitation of traditional feedback systems is the heavy reliance on manual input and review. Senders often have to locate the right channel, fill out forms, and developers must later sort and categorize these reports. This not only increases Sender effort but also slows down processing and response [9]. Moreover, static forms don't adapt to different Senders or dynamic situations.

UFRM transforms this by using context-aware automation at both design-time and runtime. It identifies the best feedback method based on system and Sender conditions. For instance, if the system detects that a Sender is in a stressful scenario (e.g., driving), it activates simplified input like voice or auto-logging, reducing interruption. Internally, the system classifies, prioritizes, and routes feedback automatically, using indicators such as severity or topic, which addresses the challenge of data overload and inconsistency mentioned in `Section 2.4.1` [10]. These features not only improve Sender participation but also reduce manual workload for feedback receivers [45].

5.1.3 Sensitivity to Dynamic Scenarios

Traditional systems treat all feedback contexts equally, assuming Senders are always able to type, focus, and provide structured responses. But in real-world dynamic scenarios, like medical calls, outdoor navigation, or time-critical alerts, Senders face constraints such as cognitive overload, poor connectivity, or safety concerns. This often leads to skipped or low-quality feedback, as shown in `Sections 2.4.3` and `2.4.4`.

UFRM is specifically designed to adapt to such dynamic conditions. It recognizes environmental and mental factors such as time pressure, network issues, or system instability, and adjusts its strategy accordingly. For example, in poor connectivity zones, UFRM stores feedback offline for later upload. During multitasking, it enables implicit feedback collection (e.g., from sensor data). This ensures that valuable Sender insight is not lost just because conditions are suboptimal [48]. These adaptive strategies make UFRM highly suited for real-world mobile and safety-critical situations [49].

5.1.4 Cross-Layer Integration and Adaptive Flow

Based on the interviews conducted in this study, most traditional systems separate feedback collection, analysis, and response into isolated tools or departments. This causes broken feedback loops, e.g., Senders never receive follow-up, and developers cannot trace context across stages. It also increases the risk of losing feedback relevance and reduces Sender motivation to engage, as noted in Section 2.4.4.

UFRM provides a fully connected feedback lifecycle. Each stage, collection, preprocessing, routing, analysis, and response, is part of a unified structure. Decisions in earlier layers affect downstream logic. For example, the feedback input mode (voice, form, log) affects how it is parsed and routed[49]. Response mechanisms are also linked: after the issue is processed, Senders receive automatic confirmations or updates, closing the loop and building trust. Studies confirm that systems which offer acknowledgment and transparency gain more sustained engagement from Senders [13].

UFRM directly addresses the shortcomings of traditional systems by enabling real-time operation, automation, dynamic adaptability, and continuous communication. These enhancements are essential for maintaining effective user-system feedback loops in complex, changing, and high-stakes conditions.

5.2 Answers to the Research Questions

This section answers the research questions proposed in `Chapter 1` (see `Table 1.1`), based on the findings from our thematic analysis and the validation of the UFRM model.

RQ1: What are the key components and characteristics of the User Feedback Reference Model (UFRM) for collecting and processing user feedback in dynamic scenarios?

The UFRM was designed to structure the full lifecycle of feedback handling in dynamic situations. It is built on four interrelated layers:

- **L1 – Dynamic Scenario Identification**: Detects time pressure, system instability, or task interference to determine whether feedback needs special handling.
- **L2 – Feedback Collection**: Matches the Sender's current context and effort level with the most suitable feedback mechanism, supporting options like voice, one-tap input, or automated logs.
- **L3 – Feedback Processing**: Organizes feedback into structured units, categorizes and prioritizes it for routing.
- **L4 – Feedback Response**: Sends a meaningful and timely acknowledgment back to the sender, closing the loop.

The architecture of UFRM reflects these requirements by combining context-awareness, flexibility in input formats, and structured processing. Each class in the UML diagram (see `Section 4.4`) is designed to represent a distinct stage in the **feedback flow**, and the process model shows how the **entity classes** interact across all four layers.

RQ1.1: What are the characteristics of dynamic scenarios?

Dynamic scenarios are set of conditions where Senders face time-sensitive, environmental constraints, system instability, or cognitively demanding conditions while interacting with digital feedback systems. Thematic analysis of the interviews revealed four key challenges for these Dynamic Scenarios:

- Feedback becomes harder to submit when **time pressure** increases.
- **Network connetion** and **environmental noise** hinder real-time feedback.
- **System instability** reduces user trust and motivation to provide input.
- Senders may skip or delay feedback during **high cognitive workload** (e.g., while driving or attending medical calls).

These insights motivate exploiting feedback from the sender's environmental context to detect scene conditions automatically. UFRM's **L1 - Dynamic Scenario Recognition layer** (see `Section 4.2.1`) enables the system to adapt its feedback mechanisms based on factors such as time pressure and environmental noise.

RQ1.2: What are the preferences and challenges of feedback senders for providing feedback in dynamic scenarios?

The findings for **RQ 1.2** come from `Section 4.2`, where we analyzed the interviews with feedback senders, and from `Section 4.3`, where we summarized the key insights (mainly from **Themes 1 to 3**). The results show that senders want feedback methods that are quick, easy to use, and fit the current situation. But they often face problems like time pressure, stress, and not knowing how or when to give feedback. These insights help us understand how to design a feedback system that works better for Senders in dynamic conditions. Here is the summary:

Main Preferences:

- Want low-effort ways to give feedback (e.g., voice input, one-tap survey).
- Prefer to give feedback during or right after the task.
- Expect quick confirmation or updates from the system.
- Like mechanisms that match their current context (e.g., driving, emergency).
- Value feedback that feels meaningful and is taken seriously.

Main Challenges:

- Too much time pressure to report feedback.
- Difficult to focus or type during high-stress tasks.
- Unsure when or how to give feedback.
- Feel like feedback goes into a "black box" with no response.
- Find it's hard to give feedback if the system crashes or performs poorly.

RQ1.3: What are the preferences and challenges of feedback receivers for processing user feedback in dynamic scenarios?

Similarly, the results for **RQ 1.3** are based on Section `Section 4.2`, where we reviewed interviews with feedback receivers, and Section `Section 4.3`, where we summarized their needs and problems (mainly from **Themes 4 to 6**). Receivers want feedback that is clear, well-organized, and easy to act on. But they often get reports that are messy, vague, or missing key information. They also find it hard to respond to Senders. These insights show why feedback systems should support easier processing, better structure, and simple ways to send responses back to Senders. Here is the summary:

Main Preferences:

- Want structured and clear feedback content.
- Prefer feedback with useful metadata (e.g., time, location, system state).
- Need help classifying and prioritizing feedback.
- Expect feedback tools to support team workflows (e.g., task routing).
- Appreciate a system that helps close the feedback loop with Senders.

Main Challenges:

- Receive vague, emotional, or unstructured input.
- Spend time manually organizing and labeling feedback.
- Miss important issues due to unclear or delayed reports.
- Have limited capacity to reply to every sender.
- Find it hard to know which feedback is urgent or impactful.

RQ2: To what extent is the proposed User Feedback Reference Model (UFRM) applicable in real-world dynamic scenarios?

To evaluate the applicability of UFRM, in the `Section 4.5`, we followed the validation steps proposed in the Design Science Research Methodology (DSRM) [11], specifically focusing on demonstration and evaluation using three use cases inspired by real-world dynamic contexts. Each use case simulated a situation where Senders needed to submit feedback under conditions of time pressure, task interference, or system instability.

In the **demonstration** phase, we mapped each scenario step-by-step to the four layers of UFRM, showing how the model detects the context, collects feedback through a suitable mechanism, processes it with structured logic, and generates a response. In the **evaluation** phase, we assessed whether UFRM fulfilled its design goals (see `Section 1.4.1`), including ease of use, feedback clarity, and responsiveness. The results show that UFRM is capable of adapting to high-pressure or unstable conditions, supports multimodal and low-effort feedback collection, and ensures that feedback is processed and routed effectively. The system also generates responses that are meaningful and timely, thereby supporting both the sender's and receiver's needs.

5.3 Threats to Validity

This section discusses the threats to validity that may affect our study's results and conclusions. We follow Runeson and Höst's guidelines [50] for case study research in software engineering in four aspects: construct, internal, external, and reliability validity.

Construct Validity

Construct validity concerns whether the concepts we studied reflect what we intended to investigate [50]. This means checking if the interview questions and thematic analysis captured how Senders experience feedback in dynamic scenarios. We addressed this by designing interview guides that were grounded in the feedback process, and by reviewing all codes collaboratively between the two of us. However, there is a chance that terms like "Sender effort" or "system instability" were understood differently by different participants. We used a six-step thematic method to reduce this risk and continuously aligned our codes with the research questions.

Internal Validity

When the researcher studies the relationship between factors, internal validity is important, and the researcher wants to understand if one affects the other [50]. In our study, we found what dynamic conditions affect user feedback behavior. We found consistent patterns indicating that these conditions reduce Senders' ability or willingness to provide feedback. We did not consider Sender emotions or affective states as uncontrolled confounders. Instead, we explicitly considered emotions and cognitive load as part of the dynamic scenario and incorporated them into our thematic analysis. To ensure internal validity, we used multiple interview perspectives and cross-checking during the coding process to strengthen our interpretation of causal patterns.

External Validity

External validity examines whether the findings apply to situations outside the scope of the study [50]. While we did not use a statistically selected sample, our goal was to gain insights that apply to similar scenarios. To test its flexibility, we applied the UFRM to three real-world inspired use cases: navigation, autonomous driving, and healthcare. Each use case was built based on common themes from Sender interviews. We do not claim that the model has perfect generalization capabilities, but the model is designed to support analytical generalization. This means that it may apply to other dynamic conditions with similar feedback challenges.

Reliability

Reliability refers to whether the results of the study would be the same if done by someone else [50]. In this study, we documented each step clearly. All interview transcripts were coded using a shared scheme, and then we reviewed the data and discussed the findings together. The thematic steps were transparent and repeatable. Although interpretations may vary slightly, we believe our process is consistent and reliable.

5.4 Informed Consent

In line with ethical research practices, all participants involved in this study were fully informed about the purpose, scope, and voluntary nature of their participation before the interview sessions took place. At the start of each interview, the researcher explained the study's goals, the interview's structure, and how the feedback would be used, specifically for academic research aimed at developing and validating the UFRM. Participants were made aware that their answers would remain anonymous and confidential, and that the results would be reported in an aggregated format, without linking responses to individual identities.

We obtained verbal consent from each participant and recorded it. In cases where interviewees had additional questions about the study or data usage, the researcher provided further clarification regarding the storage of audio recordings and transcriptions and the right to withdraw at any point. No personally identifiable information (such as names, job titles, or organizational affiliations) was collected, ensuring participant privacy. Data were stored securely and used only for qualitative analysis and model validation. This approach complies with ethical guidelines for academic research and ensures transparency and respect for participant autonomy.

5.5 Usage of Generative AI in This Thesis

During the preparation and writing of this thesis, Generative AI tools, most notably ChatGPT by OpenAI and Grok, were used in a limited and clearly defined manner. These tools served primarily to support the authors in generating early drafts of explanatory text, improving phrasing in English, and structuring LaTeX code for formatting tables, figures, and references. At no point were generative AI systems used to write original academic content, perform qualitative analysis, or generate interpretations of interview data.

The authors conducted all critical parts of the thesis, such as the formulation of research questions, development of the UFRM, thematic analysis of interview transcripts, model validation, and synthesis of findings. The AI-supported components were carefully reviewed and edited to ensure academic integrity and alignment with the research context. The use of AI was fully under human supervision and never replaced any cognitive, evaluative, or creative aspects of the thesis work. Its purpose was to enhance efficiency during the editing process and maintain clarity in writing without compromising academic standards. This responsible use reflects growing trends in academia regarding the use of AI tools as supportive instruments rather than content generators.

5.6 Future Works

Although the UFRM proposed in this study has achieved results in structural design and simulation verification, several important directions still deserve further exploration. Here are two suggestions for future research work.

First, the verification method adopted in this study mainly relies on the demonstration of simulated scenarios (use case-based demonstration). Typical dynamic feedback scenarios are constructed through interview data, and the model process and class structure are combined for analysis. However, the model has not yet been deployed and applied in a real system environment. Future research should consider integrating UFRM into **actual running software systems**, such as navigation applications, telemedicine platforms, and in-vehicle systems, and interacting with real Senders. By observing the actual feedback behavior of Senders in dynamic scenarios and combining system log data and Sender operation records, we can further verify the stability and Sender acceptance of UFRM in actual operation.

Second, UFRM currently relies mainly on rule-driven logic for scenario identification and mechanism recommendation, i.e., selecting feedback processes based on preset scenario attributes and Sender status. Although this approach is structurally clear and controllable, fixed rules may not be able to fully cope with various Sender behaviors and feedback intentions in complex and frequently changing real environments. Therefore, future research can try to introduce **data-driven methods**, especially machine learning-based model training mechanisms, so that the system can continuously learn and adjust based on historical interaction data and user feedback content. For example, by training models with information such as Sender portraits, context history, and feedback processing results, dynamic recommendations for feedback mechanisms, fine judgments on Sender status, and automatic generation of response content can be achieved, thereby further improving the intelligence, adaptability, and cross-scenario generalization capabilities of UFRM.

5.7 Conclusion

This paper explores how to better manage user feedback in dynamic scenarios, such as when time pressure, environmental constraints, system instability, and task interruptions make traditional feedback mechanisms difficult to use. Through thematic analysis of interviews with feedback senders and receivers, we identified key challenges and preferences, which guided the design of the User Feedback Reference Model (UFRM). The model was designed to support low-effort, context-aware feedback interactions and ensure that Senders and system receivers benefit from clearer, more actionable input.

UFRM consists of four layers: dynamic scenario identification, feedback collection, processing, and response. Each layer addresses specific needs identified in our data. To validate the model, we applied it to three real-world-inspired use cases in smart navigation, autonomous driving, and digital healthcare. These examples showed how UFRM works step by step, automatically activating suitable feedback mechanisms and organizing feedback for efficient processing and timely response. The model proved adaptable and practical in dynamic Scenarios.

Our study contributes a structured solution for managing user feedback under dynamic conditions. UFRM helps reduce Senders' cognitive burden while also improving organizations' processing. While the model has been validated through use case simulations, future work can explore integration in real systems, broader user testing, and refinement across more domains. UFRM offers a starting point for building smarter, more responsive feedback systems that work in real time.

Bibliography

[1] F. Fotrousi, "Combining user feedback and monitoring data to support evidence-based software evolution," Ph.D. dissertation, Blekinge Institute of Technology, 2020.

[2] A. L. Cavaye, "User participation in system development revisited," *Information & Management*, vol. 28, no. 5, pp. 311–323, 1994.

[3] S. van Oordt and E. Guzman, "On the role of user feedback in software evolution: A practitioners' perspective," in *Proceedings of the 29th IEEE International Requirements Engineering Conference (RE)*. IEEE Computer Society, 2021, pp. 221–232.

[4] J. O. Johanssen, A. Kleebaum, B. Bruegge, and B. Paech, "How do practitioners capture and utilize user feedback during continuous software engineering?" in *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE, 2019, pp. 153–164.

[5] W. M. et al., "On the automated processing of user feedback," *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 1–12, 2024.

[6] E. Yigitbas, A. Hottung, and S. M. Rojas, "Context- and data-driven satisfaction analysis of user interface adaptations based on instant user feedback," in *Proceedings of the 27th International Conference on Human-Computer Interaction (HCII)*, 2019, pp. 621–633.

[7] I. Morales-Ramirez, A. Perini, and R. Guizzardi, "Providing foundation for user feedback concepts by extending a communication ontology," in *Conceptual Modeling – ER 2014*, ser. Lecture Notes in Computer Science, A. Ghose and M. Grossniklaus, Eds., vol. 8824. Springer, 2014, pp. 305–312.

[8] S. R. Methods, "Semi-structured interview," in *The SAGE Encyclopedia of Social Science Research Methods*. SAGE Publications, 2024. [Online]. Available: https://methods.sagepub.com/ency/edvol/the-sage-encyclopedia-of-social-science-research-methods/chpt/semistructured-interview

[9] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006. [Online]. Available: https://www.researchgate.net/publication/235356393_Using_thematic_analysis_in_psychology

[10] M. Naeem, W. Ozuem, K. Howell, and S. Ranfagni, "A step-by-step process of the-

matic analysis to develop a conceptual model in qualitative research," *Qualitative Market Research: An International Journal*, vol. 26, no. 1, pp. 123–146, 2023.

[11] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007.

[12] L. G. Carreño and K. Winbladh, "Analysis of user comments: an approach for software requirements evolution," in *Proceedings of the 2013 International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 582–591.

[13] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE, 2014, pp. 153–162.

[14] M. Bano and D. Zowghi, "A systematic review on the relationship between user involvement and system success," *Information and Software Technology*, vol. 58, pp. 148–169, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0950584914001221

[15] Z. S. Li, N. N. Arony, K. Devathasan, M. Sihag, N. A. Ernst, and D. E. Damian, "Unveiling the life cycle of user feedback: Best practices from software practitioners," in *Proceedings of the 46th International Conference on Software Engineering (ICSE)*, 2024, p. to appear.

[16] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe, "Toward Data-Driven Requirements Engineering," *IEEE Software*, vol. 33, no. 1, pp. 48–54, 2016.

[17] J. Dabrowski, E. Letier, A. Perini, and A. Susi, "Analysing app reviews for software engineering: a systematic literature review," *Empirical Software Engineering*, vol. 27, no. 2, p. 43, 2022.

[18] A. J. Ko, M. J. Lee, V. Ferrari, S. Ip, and C. Tran, "A case study of post-deployment user feedback triage," in *Proc. of the 4th Int. Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2011, pp. 1–8.

[19] S. Chattopadhyay, N. Dalal, M. Burnett, M. Gerosa, and A. Sarma, "Cognitive biases in software development," *Communications of the ACM*, vol. 65, no. 4, pp. 115–122, 2022.

[20] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requirements Engineering*, vol. 21, no. 3, pp. 311–331, 2016.

[21] R. Atterer, M. Wnuk, and A. Schmidt, "Knowing the user's every move: user activity tracking for website usability evaluation and implicit interaction," in *Proceedings of the 15th International Conference on World Wide Web (WWW)*. ACM, 2006, pp. 203–212. [Online]. Available: https://dl.acm.org/doi/10.1145/1135777.1135811

[22] K. R. Lakhani and E. von Hippel, "How open source software works: "free" user-to-user assistance," *Research Policy*, vol. 32, no. 6, pp. 923–943, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0048733302000951

[23] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users

complain about?" *IEEE Software*, vol. 32, no. 3, pp. 70–77, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/6899235

[24] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, and J. A. Landay, "Myexperience: A system for in situ tracing and capturing of user feedback on mobile phones," in *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services*, 2007, pp. 57–70.

[25] M. White, M. Linares-Vásquez, P. Johnson, C. Bernal-Cárdenas, and D. Poshyvanyk, "Generating reproducible and replayable bug reports from android application crashes," in *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension (ICPC)*. IEEE, 2015, pp. 48–59. [Online]. Available: https://ieeexplore.ieee.org/document/7181447

[26] J. Linaker, S. M. Sulaman, R. M. de Mello, and M. Höst, "Guidelines for conducting surveys in software engineering," *Empirical Software Engineering*, vol. 20, pp. 1442–1474, 2015.

[27] A. J. Bingham, "From data management to actionable findings: A five-phase process of qualitative data analysis," *International Journal of Qualitative Methods*, vol. 22, pp. 1–16, 2023.

[28] D. Pagano, "Portneuf - a framework for continuous user involvement," in *IEEE Transactions on Software Engineering*, vol. 39, no. 4, 2013, pp. 469–487.

[29] T. Zimmermann, R. Premraj, and N. Bettenburg, "What makes a good bug report?" *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618–643, 2010.

[30] M. Oriol, M. Stade, F. Fotrousi, S. Nadal, J. Varga, X. Franch, D. Méndez, and N. Koch, "FAME: Supporting continuous requirements elicitation by combining user feedback and monitoring," in *26th IEEE International Requirements Engineering Conference (RE)*. IEEE, 2018, pp. 217–227.

[31] M. Mehraban, "Instant feedback loops – for short feedback loops and early quality assurance," Ph.D. dissertation, Chalmers University of Technology, 2016.

[32] J. Ruiz and M. Snoeck, "Automatic feedback generation for supporting user interface design," *Journal of Systems and Software*, vol. 178, p. 110915, 2021.

[33] M. A. Hadi and F. H. Fard, "Evaluating pre-trained models for user feedback analysis in software engineering: A study on classification of app-reviews," *Empirical Software Engineering*, vol. 28, pp. 1–24, 2023.

[34] N. Chen, J. Wu, X. Liu, H. Hu, Y. Sun, S. C. H. Hoi, and X. Xing, "AR-Miner: Mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering (ICSE)*. ACM, 2014, pp. 767–778.

[35] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," *21st IEEE International Requirements Engineering Conference (RE)*, pp. 125–134, 2013.

[36] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[37] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0164121215001357

[38] Z. S. Li, N. N. Arony, and D. Damian, "Uncovering patterns in users' ethical concerns about software," *IEEE Transactions on Software Engineering*, pp. 1–16, 2024.

[39] L. López, A. Ahbervé, and A. Bagnato, "Qfl: Data-driven feedback loop to manage quality in agile development," *Journal of Software: Evolution and Process*, vol. 33, no. 5, p. e2345, 2021.

[40] M. Anders, M. Obaidi, A. Specht, and B. Paech, "What can be concluded from user feedback? an empirical study," in *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. IEEE, 2023.

[41] S. van Oordt and E. Guzman, "On the role of user feedback in software evolution: A practitioners' perspective," *Empirical Software Engineering*, vol. 26, pp. 1–35, 2021.

[42] S. A. Neslin *et al.*, "Challenges and opportunities in multichannel customer management," *Journal of Service Research*, vol. 9, no. 2, pp. 95–112, 2006.

[43] A. Tkalich, E. Klotins, and A. Barbala, "User feedback in continuous software engineering: revealing the state-of-practice," *Empirical Software Engineering*, vol. 30, no. 3, p. 79, 2025.

[44] B. F. E. Sharkawy and F. Meawad, "Instant feedback using mobile messaging technologies," *Mobile Information Systems*, vol. 5, no. 4, pp. 253–265, 2009.

[45] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? on automatically classifying app reviews," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*. Ottawa, ON, Canada: IEEE, 2015, pp. 116–125.

[46] N. Seyff, G. Ollmann, and M. Bortenschlager, "AppEcho: a user-driven, in situ feedback approach for mobile platforms and applications," in *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems (MOBILESoft 2014)*. Hyderabad, India: ACM, 2014, pp. 99–108.

[47] L. A. Palinkas, S. M. Horwitz, C. A. Green, J. P. Wisdom, N. Duan, and K. Hoagwood, "Purposeful sampling for qualitative data collection and analysis in mixed method implementation research," *Administration and Policy in Mental Health and Mental Health Services Research*, vol. 42, no. 5, pp. 533–544, September 2015.

[48] P. Fusch and L. Ness, "Are we there yet? data saturation in qualitative research," *The Qualitative Report*, vol. 20, pp. 1408–1416, 2015. [Online]. Available: https://nsuworks.nova.edu/tqr/vol20/iss9/3/

[49] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *Proceedings of the 21st IEEE International Requirements Engineering Conference (RE)*. IEEE, 2013, pp. 125–134. [Online]. Available: https://ieeexplore.ieee.org/document/6636712

[50] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research

in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

A

Interviews with feedback receivers

Opening

"Thank you for taking the time to participate in this interview. This study is part of our master's thesis at Chalmers University of Technology, focusing on how companies collect and process user feedback to improve their products, especially in fast-changing (dynamic) scenarios. Your insights will be greatly invaluable and help us refine our User Feedback Reference Model to address practical solutions for industry challenges. The interview will take approximately 40-60 minutes. Your responses will remain anonymous and confidential while collecting, analysing and storing, and they will be used solely for research purposes. With your permission, we would like to record this session for accuracy."

Hoda Sheikh & Haoru Sui

A.1 Demographic Information

1.1 General Information:

Can you briefly introduce yourself, your role in the company, and how large is your company? Also, explain what product you work on and what is the approximate number of active users?

1.2 Experience with User Feedback:

How long have you been working in this company, and what has been your experience with user feedback so far?

A.2 User Feedback Collection

2.1 Feedback Collection Methods

How do you usually collect user feedback? Do you ask users directly (explicitly) for feedback (e.g., surveys, ratings) or do you collect data indirectly (implicitly) by monitoring users' behavior (e.g., interactions, logs)? Have you tried combining both methods?

2.2 Feedback Formats In what format do you usually receive feedback (e.g., text, audio, screenshots, logs)?

2.3 Feedback Channels

Through which channels do users typically provide feedback? (e.g., in-app reporting, support tickets, emails, telephone, live chat, forums, app store reviews)

2.4 Types of Feedback / Feedback Context

What are the most common types of feedback you receive? (e.g., bug reports, feature requests, usability issues, performance complaints)

2.5 Challenges in Collecting Feedback

What difficulties do you face when gathering user feedback? How do you handle these challenges? (e.g., feedback overload, low quality, conflicting feedback, lack of structure, irrelevant/fake reports, missed/delayed feedback)

A.3 Feedback Processing & Analysis

3.1 Ensuring Useful Feedback / Feedback Effectiveness

What characteristics (specification, metrics, attributes) make user feedback useful (effective)? How do you define good user feedback?

3.2 Feedback Processing Workflow

How do you process (categorize, analyze, act on) the feedback you collect?

3.3 Tools and Methods for Analysis Do you use any specific tools or software to analyze and manage user feedback? If not, how do you handle it manually?

3.4 Prioritization Strategies

How do you categorize and prioritize the feedback you receive?/ How do you determine which feedback to address first? (e.g., based on frequency, severity, impact on users, business priorities)

3.5 Feedback Routing

How do you decide which team or department should handle each type of feedback?

3.6 Action on Feedback

What do you do with the feedback you received? How do you use it?

3.7 User Follow-up and Communication

Do you inform users about how their feedback is processed? If yes, how do you communicate updates back to them?

3.8 Challenges in Processing Feedback

What obstacles do you encounter when analyzing feedback, and what strategies do you use to manage them?

A.4 Dynamic Scenarios

4.1 User Feedback in Dynamic Scenarios

Have you encountered situations where users provided feedback in fast-changing environments, such as during driving, online meetings, deliveries, or medical emergencies? Can you share an example?

4.2 Characteristics of Dynamic Scenarios

What are the key considerations that make feedback collection difficult in these situations? (e.g., time pressure, distractions, urgency, technical constraints)

4.3 Effective Feedback Collection in Dynamic Scenarios

Have you implemented or considered any effective methods to collect feedback in these scenarios? (e.g., voice commands, automated logs, simplified reporting buttons)

4.4 Timing of Feedback Submission

Do users usually provide feedback during or after the dynamic activity? How does this impact the quality of feedback?

4.5 Instant Feedback Collection

Does your system support instant feedback submission, anytime and anywhere? Do you think this would be useful in dynamic scenarios?

4.6 Feedback Handling in Dynamic Scenarios

How is collecting and analyzing feedback in dynamic scenarios different from in normal situations?

A.5 Closing Questions

5.1 Improving Feedback Collection and Analysis

If you could improve your feedback system, what features or methods would you consider adding?

5.2 Additional Insights

Is there anything else about user feedback collection and analysis that you think we should explore further?

5.3 Follow-up & Networking

Would you be open to follow-up discussions and receive our research results? Can you recommend any professionals who might provide further insights?

B

Interviews with feedback senders

Opening

"Thank you for taking the time to participate in this interview. This study is part of our master's thesis at Chalmers University of Technology, focusing on how end users give feedback and how companies collect and process user feedback to improve their products, especially in fast-changing (dynamic) situations. Your insights will be greatly invaluable and help us refine our User Feedback Reference Model (UFRM) to address practical solutions for industry challenges. The interview will take approximately 10-15 minutes of your valuable time. Your responses will remain anonymous and confidential while collecting, analysing and storing, and they will be used solely for research purposes. With your permission, we would like to record this session for accuracy." Hoda Sheikh Haoru Sui

B.1 General Experience with Giving Feedback

1. Experience:

Do you have any experience of giving feedback on a software product? On your phone or on a website? Which system/ app? (Can you explain it a little bit or can you give your examples) (RQ1.2)

2. Frequency (when) :

Do you remember when was your last experience? Can you say how often you usually provide feedback? (RQ1.2)

3. Feedback content (what/why):

What feedback content did you typically provide for software systems?(Why do you provide feedback?) (RQ1.2)

4. How (Which Format & Channel) :

What were your preferred typical feedback formats and channels, if you have provided any before? (RQ1.2)

B.2 Feedback in Dynamic Scenarios

5. Experience in Dynamic Scenarios(Characteristics):

Have you ever been placed in a scenario that you were in stress or unstoppable process because of time restrictions or a rapidly changing environment, and the system asked you for feedback? What do you feel? Will you give feedback in these situations? (Please describe it briefly.) (RQ1.1)

6. Prevention (Challenges):

If NOT, Why won't you give feedback in those scenarios? What are the challenges you may face in these situations? (Please describe it briefly.) (RQ1.3)

7. Motivation:

If you think it can be helpful for you to give feedback , what motivates/encourages you to provide feedback under dynamic scenarios? (RQ1.3)

8. Feedback System Features :

What do you think a good/satisfiable feedback collection system should look like under dynamic scenarios? (What can be its specifications/ Considerations/ characteristics?) (RQ1.2)

9. How To Give Feedback :

How would you prefer to submit feedback under dynamic scenarios? (In which format, content, channel?) (RQ1.2)

10. Feedback to Feedback :

How important is it for you to receive a response from the company after submitting your feedback? (How much do you prefer to receive F2F)? (RQ1.3)

11. Additions:

Do you have any additional comments or suggestions on how to improve the feedback system under dynamic scenarios? (RQ1.2)

B.3 About You (Demographic Information)

Closing question:(All info will be anonymous and confidential as we mentioned before)

12. Age: What is your age group (less than 30, 30 to 50, over 50)?

13. Gender : What is your gender (female, male, do not prefer to say)?

14. Digital Experience : How much is your Digital Experience with mobile phones, computers (beginner, skilled, proficient)

C

Entity Classes and Their Associations in UFRM

Entity Class	Description	Associated Classes and Lists
DynamicScenario	Detects and measures real-time contextual factors (such as time pressure, instability, safety, etc.) that influence user and system behavior, and triggers generation of appropriate feedback mechanisms based on the scenario type.	**TimePressure** : High, Medium, Low **SystemInstability**: High, Medium, Low **EnvironmentalNoise** : High, Medium, Low **TaskDisruption :** High, Medium, Low **SafetyLevel :** High, Medium, Low **ContextualFactors :** TimePressure, SystemInstability, environmentalNoise, taskDisruption, safetyLevel
FeedbackMechanism	Predefined set of configurations tailored for a specific dynamic scenario. It bundles design-time attributes (like format, channel, trigger, response type) and defines which entities participate in that scenario's feedback handling.	**FeedbackTrigger** : Human-Explicit, Automatic-Implicit **TriggeringTime** : During Task, After Task, Idle Time **SubmissionChannel :** Support ticket, Email, Live chat, Chat bot, Customer servise phone, Internal reporting, Appstore review, website, In-app report, ... **DataFormat** : Text, Images, Audio, Video, Sensor Data, Metrics-track... **ProcessMethod** : Manual, Automaive, Hybrid **ResponseTime** : Immediate, Delayed **ResponseType** : Acknowledge, Resolve, Inform,... **ActiveFBMechanism**
ActiveFeedbackMechanism	Runtime version of a selected FeedbackMechanism, personalized and activated for a specific FeedbackSender based on their profile and contextual factors. It manages the logic of how that sender can interact with the system.	**FBData**
FeedbackSender	The human or machine agent that provides feedback. Its characteristics (such as effort level, trust, mood, and identity) affect how the system adjusts and delivers feedback mechanisms in real time.	**SenderIdentityInfo** : Role, Gender, Age Gourp, Experience-level, ... **SenderType** : Human, Machine **MotivationLevel :** High, Medium, Low **Mood** : Positive, Neutral, Negative **EffortLevel** : High, Medium, Low **TrustLevel** : High, Medium, Low **SenderFactors** : SenderType, MotivationLevel, Mood , EffortLevel , TrustLevel **ResponseContent**
FeedbackData	The raw input generated by the sender (explicit or implicit), containing the actual content of the feedback. It is enriched with metadata (e.g., time, location, device) and forwarded for processing.	**FBDataContent** : Question, Answer **Metadata** : Device, Location, ...
FeedbackDataPackage	A structured grouping of multiple feedback data entries (typically submitted together or by one sender). It serves as the processing unit for pre-processing and routing.	**List <FBData>**
FeedbackProcessor	Handles cleaning, categorizing, and prioritizing the feedback data package. It identifies limitations, transforms feedback into structured content, and prepares it for decision-making by the receiver.	**List <DataLimitation>** : Unclear, Incomplete, Irrelevant, Redundant, Conflicting, Unstructured, Delayed, Missed, ... **DataCategoryTopic** : Errors, Bugs, Feature Requests, Suggestions, Usability Issues, Performance Complaints, ... **DataPriority :** Severity, Potential danger, Business impact, Frequency, Safety and Risk, ...
FeedbackReceiver	The final decision-maker (human or machine) that interprets processed feedback and acts on it—either by resolving, escalating, or rejecting it. It also generates a response for the sender.	**ReceiverIdentityInfo** : Role, Gender, Age Gourp, Experience-level, ... **ProcessRoutingInfo** : User, Request Clarification, Bug Fix or Patch, Rerouting to Specialist, Schedule for Sprint, Reject Feedback, ... **Response**
FeedbackResponse	The output message sent back to the sender. It contains the result of the feedback handling process (e.g., acknowledgment, resolution), completing the feedback loop and helping build user trust.	**ResponseContent**

Figure C.1: Entity Classes in UFRM

Entity Class	Associated Class / List	Type	Description	Values
DynamicScenario	TimePressure	Enumeration Class	Represents the level of time pressure in the scenario, indicating how urgently tasks need to be completed	**High:** A very tight deadline to submit feedback **Medium:** A manageable deadline to submit feedback **Low:** A relaxed deadline to submit feedback
	SystemInstability	Enumeration Class	Represents the stability of the system, based on the frequency of occurrences of errors or failures	**High**: High instability, with severe system errors occurring often **Medium**: Medium instability, with moderate system errors occurring occasionally **Low**: Low instability, with minimal or no system errors
	EnvironmentalNoise	Enumeration Class	Represents the level of environmental factors that might disturb feedback delivery	**High**: High noise level that affecting feedback submission **Medium**: Moderate noise level that affecting feedback submission **Low**: Low noise level that affecting feedback submission
	TaskDisruption	Enumeration Class	Represents the frequency of interruptions that disrupt tasks in the scenario.	**High**: Frequent disruptions **Medium**: Occasional disruptions **Low**: Minimal or no disruptions
	SafetyLevel	Enumeration Class	Represents the safety level of the scenario based on the number of safety risks or alerts	**High**: No safety risks **Medium**: Minor safety risks **Low**: Significant safety risks
	ContextualFactors	Independent Class	A class that holds the contextual factors of a DynamicScenario, used to encapsulate measurement results	**timePressure**: The level of time pressure **systemInstability**: The system stability level **environmentalNoise**: The environmental noise level **taskDisruption**: The task disruption frequency **safetyLevel**: The safety level
FeedbackMechanism	RequestingTime	Enumeration Class	Represents the timing of when feedback is requested in the dynamic scenario	**During Task:** Feedback is requested while the task is in progress **After Task:** Feedback is requested after the task is completed **Idle Time:** Feedback is requested during idle periods
	FeedbackTrigger	Enumeration Class	Represents the method by which feedback is initiated	**Human-Explicit:** Feedback is explicitly initiated by a human user **Automatic-Implicit:** Feedback is automatically triggered by the system
	SubmissionChannel	Enumeration Class	Represents the channel through which feedback is submitted	**Support Ticket:** Feedback submitted via a support ticket system **Email:** Feedback submitted via email **Live Chat:** Feedback submitted through a live chat interface **Chat Bot:** Feedback submitted via a chatbot **Customer Service Phone:** Feedback submitted through a customer service phone call **Internal Reporting:** Feedback submitted via an internal reporting system **Appstore Review:** Feedback submitted as an app store review **Website:** Feedback submitted through a website form **In-App Report:** Feedback submitted directly within the app **Others**
	DataFormat	Enumeration Class	Represents the format of the feedback data collected	**Text:** Feedback in text form (e.g., written comments) **Images:** Feedback in image form (e.g., screenshots) **Audio:** Feedback in audio form (e.g., voice recordings) **Video:** Feedback in video form (e.g., screen recordings) **Sensor Data:** Feedback from sensors (e.g., device sensor readings) **Metrics-Track:** Feedback based on tracked metrics (e.g., usage statistics) **Others**
	ProcessMethod	Enumeration Class	Represents the method used to process the feedback	**Manual:** Feedback is processed manually by a human **Automated:** Feedback is processed automatically by a system (e.g., using AI or scripts) **Hybrid:** Feedback is processed using a combination of manual and automated methods
	ResponseType	Enumeration Class	Represents the type of response provided to the feedback	**Acknowledge:** A simple acknowledgment of the feedback **Resolve:** A response that resolves the issue **Inform:** A response that provides information **Others**
FBSender	SenderIdentityInfo	Independent Class	Contains the identity details of the sender	**Role:** The sender's role in the system (e.g., Admin, User) **Gender:** The sender's gender (e.g., Male, Female) **Age Group:** The sender's age group (e.g., Youth, Adult) **Experience-level:** The sender's experience level (e.g., Beginner, Expert) **Others**
	SenderType	Enumeration Class	Represents the type of sender	**Human:** The sender is a human user **Machine:** The sender is a machine or automated system
	MotivationLevel	Enumeration Class	Represents the sender's motivation level for providing feedback	**High:** The sender is highly motivated **Medium:** The sender has moderate motivation **Low:** The sender has low motivation
	Mood	Enumeration Class	Represents the sender's emotional state when providing feedback	**Positive:** The sender is in a positive mood **Neutral:** The sender is in a neutral mood **Negative:** The sender is in a negative mood
	EffortLevel	Enumeration Class	Represents the level of effort the sender puts into providing feedback	**High:** The sender puts in high effort **Medium:** The sender puts in medium effort **Low:** The sender puts in low effort
	TrustLevel	Enumeration Class	Represents the sender's trust level in the system	**High:** The sender has high trust **Medium:** The sender has medium trust **Low:** The sender has low trust
	SenderFactors	Independent Class	Contains the factors describing the sender's characteristics	**SenderType:** The type of sender **MotivationLevel:** The sender's motivation level **Mood:** The sender's mood **EffortLevel:** The sender's effort level **TrustLevel:** The sender's trust level
FBData	FBDataContent	Independent Class	Contains the content of the feedback provided by the sender	**Question:** The question asked by the feedback sender **Answer:** The answer provided by the feedback sender
	Metadata	Independent Class	Contains additional metadata about the feedback data	**Device:** The device used to submit the feedback **Location:** The location where the feedback was submitted **Others**
FeedbackDataPackage	feedbackDataList	List	Represents a collection of FBData objects for each feedback sender	Contains multiple **FBData** entries
FBProcessor	DataLimitation	List	Represents a limitation identified in the feedback data (can contain multiple values in a List)	**Unclear:** Feedback is not clear or understandable **Incomplete:** Feedback lacks necessary details **Irrelevant:** Feedback is not relevant to the context **Redundant:** Feedback repeats existing information **Conflicting:** Feedback contradicts other data **Unstructured:** Feedback lacks proper structure **Delayed:** Feedback was submitted late **Missed:** Feedback was not captured when expected **Others**
	DataCategoryTopic	Enumeration Class	Represents the category or topic of the feedback data	**Errors:** Feedback about errors in the system **Bugs:** Feedback about bugs or glitches **Feature Requests:** Feedback requesting new features **Suggestions:** Feedback offering suggestions for improvement **Usability Issues:** Feedback about usability problems **Performance Complaints:** Feedback about performance issues **Others**
	DataPriority	Enumeration Class	Represents the priority of the feedbackRepresents the priority of the feedback based on multiple factors	**Severity:** The severity level of the issue **Potential Danger:** The potential risk involved **Business Impact:** The impact on business operations **Frequency:** How often the issue occurs **Safety And Risk:** The safety risks associated with the issue **Others**
FeedbackReceiver	ReceiverIdentityInfo	Independent Class	Contains the identity details of the feedback receiver	**Role:** The receiver's role in the system (e.g., Admin, Support) **Gender:** The receiver's gender (e.g., Male, Female) **Age Group:** The receiver's age group (e.g., Youth, Adult) **Experience-level:** The receiver's experience level (e.g., Beginner, Expert) **Others**
	ProcessRoutingInfo	Enumeration Class	Represents the routing action for processing feedback	**User:** Routes feedback to the user for further action **Request Clarification:** Requests clarification from the sender **Bug Fix or Patch:** Routes feedback for bug fixing or patching **Rerouting to Specialist:** Routes feedback to a specialist for handling **Schedule for Sprint:** Schedules feedback for a future sprint **Reject Feedback:** Rejects the feedback if not actionable **Others**

Figure C.2: Associated Classes and Lists with Entity Classes

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY