A Framework for Modelling and Utilization of Users' Feedback for Software Systems Evolution

Nada Hany Hassan Sherief

A thesis submitted in partial fulfilment of the requirements of Bournemouth University for the degree of Doctor of Philosophy

July 2017

Bournemouth University

Copyright

Abstract

Most software systems operate within a complex and variable context. This poses a challenge for the requirements engineering of their software systems mainly to ensure those requirements keep pace with the changing context. To cater for such volatility, users' feedback about software, while it's in use, is a powerful tool that enables the capturing and communication of a richer and updated knowledge on how they view the software. Users understand the software as a means to meet their requirements and needs, thus, giving them a voice in the continuous evaluation of software would naturally fit this level of abstraction. This contributes in identifying problems in the software, modifying existing requirements or requesting new additional requirements leading to better users' acceptance of the software.

The traditional approach to users' feedback, which is based on data mining and text analysis, is often limited, partly due to the ad-hoc nature of users' feedback and, also, the methods used to acquire it, which are either overly relaxed, e.g. natural language and forum-like that endure a lot of impression and ambiguity, or too restrictive, e.g. ranking. To maximize the expressiveness of users' feedback and still be able to efficiently analyse it, this work proposes that feedback acquisition should be designed with that goal in mind. Hence, the need to provide foundations to develop systematic methods for the structuring and use of users' feedback is advocated in this thesis.

Also, users' evaluation feedback, while the software is in use, could be used to support engineers in accomplishing evolution tasks and taking maintenance decisions. However, there is no formalized specification that properly documents the users' problems. Besides that there is a lack of systemized methods of extracting the problems into formalized reliable specifications. In traditional methods engineers end up with huge data reporting user problems, which requires a great deal of effort and time to analyse and come with useful conclusions.

This research contributes to that aim by creating novel classifications of users' perspectives on feedback types and their constituents and how they could be structured. Furthermore, a formal systematic process for feedback acquisition and communication was developed to help engineers accomplish their tasks and to further utilize the captured feedback in extracting new/ changing requirements information. Finally, a socio-technical technical architecture is developed to illustrate how the formed workflows, methods, and models interrelate to realize the research aim.

Table of Contents

List of Figures

List of Tables

Acknowledgement

The Completion of this thesis would not have been possible without the presence and assistance of many people.

First of all, no words would really express my sincere gratitude for my first supervisor, Dr. Raian Ali, for the patient guidance, encouragement and advice he has provided throughout my PhD work. I have been extremely lucky to have a supervisor who is so enthusiastic and cared so much about my work. At many stages in the course of this research I benefited from his advice, particularly when exploring new ideas and methods. His positive outlook and confidence in my research inspired me and gave me confidence. His continuous careful editing contributed enormously in enhancing my writing, analytical, and criticizing skills. I would also like to thank my second supervisor Prof. Keith Phalp for his helpful feedback during certain stages of this PhD, and my local supervisor Assoc. Prof. Walid Abdelmoez for the support he provided during this PhD though his feedback and for helping me in the participants' recruitment of several of my PhD studies.

In addition, I'm thankful to the financial sponsor of my PhD, The Arab Academy for Science, Technology and Maritime Transport (AASTMT), in Alexandria, Egypt. Special thanks to all the management staff members of the College of Computing and Information Technology at AASTMT especially Prof. Yasser EL Sonbaty and Prof. Ayman Adel for their help and support, which have absolutely made this PhD feasible. I am also very thankful to my work colleagues for making this journey a pleasant one through their friendship, collaboration, and support.

All my gratefulness goes to my beloved husband, Muhammad Fakhry, who provided me with all the means of support and encouragement to achieve my goals. I was continually amazed by his passion about my research and by his valuable guidance as an engineer working in the software industry.

Furthermore, I would like to thank my father Prof. Hany Sherief and my mother Dr. Tahany Ahmed. All the support they have provided me over the years was the greatest gift anyone has ever given me. They taught me the value of hard work and education. Without their love, patience, support and understanding nothing would have been the way it is now. I would also like to thank my brother Adham for being there when I need him; he is a great supporter, good listener, great thinker and a true friend.

Last but not least, I would like to thank all the participants who took part in this thesis's studies for their effort, time and valuable input that helped me achieving the aim of this PhD.

Finally, I dedicate this PhD to my beloved son Tarek, whom I hope would achieve valuable goals in his life and make me and his father proud.

1. Introduction

In this chapter the overall view of the research problem and motivation is introduced. The research aims, questions, the objectives are explained, followed by a summarization for the report structure, and a list of publications that resulted from this research.

1.1 Research Overview

The derivation of Requirements Engineering (RE) research decades ago was driven by practitioners who noticed the urgent need for methodical RE in large software projects. Yet, the environment in which RE is practiced has changed dramatically since then (Jarke, Loucopoulos et al. 2011). This is due to several reasons happening at about the same time: delivery platforms are changing (mobile, cloud, social); communication and collaboration channels are being renovated (Web, mobile, social); the consumer world of technology is driving innovation; and data is opening up and overflowing out of the growing apps, devices, and sensors that organizations are deploying or are connecting to . Consequently, the field's focus and scope has moved from engineering of individual systems and components towards the generation and adaptation of software intensive ecosystems. This shift has created a strong need to understand more deeply issues that underlie current RE, and reconsider RE practices and methods to meet the new challenges. Currently, requirements management is still one of the most challenging fields in software development (Jarke, Loucopoulos et al. 2011), has the most impact on project success, and is a major issue for decision makers in enterprises.

Users' involvement along the phases of the software development life cycle is a powerful tool which enables engineers to capture and communicate a richer and updated knowledge on how users view the software (Zin and Pa 2009). In the software development life cycle, users are the main source of requirements and, thus, giving them a voice in the continuous evaluation of software, while the software is in use, would naturally fit this level of abstraction. That is, users' evaluation feedback would mainly communicate their opinion on the role of the system in meeting their requirements leading to better users' acceptance of the software. Their acceptance of the product is of high importance for market success. Therefore, in this research users are not viewed as objects of study but more as active agents within the software process, where user input (i.e. feedback) is a kind of reflection input. User's feedback while the software is in use moves away from traditional means, towards more active engagement in taking autonomous or semi-autonomous adaptation decisions or to support engineers on taking evolution and maintenance decisions.

The involvement of actual users as partners amplifies its potential and range of applications. First, it provides an ability to evaluate software while users are using it in practice. They can also characterize their evaluation by providing context description. Also, the ability to

maintain the evaluation knowledge up-to-date, as users' continuously provide their real-time feedback/judgment on the software. Furthermore, the users who will be using the system and are the most aware of its requirements and their needs will act as the actual validators of the system, which enhances the overall quality of the system. Such validation was done but often focuses on a small number of selected users to give input and feedback in requirements related activities and afterwards in user acceptance testing (Cleland-Huang, Jarke et al. 2013). This approach is expensive, time consuming, and hardly manageable. Moreover, having users to provide the input for guiding the evolution tasks and decisions eliminates the factor of uncertainty and also helps in identifying new requirements. Additionally, iteratively obtaining and processing the users' feedback will help accelerate the evolution tasks and/or maintenance process. This is particularly true for highly variable systems with a large number of alternatives, where changing requirements is the actual driver for their evolution (Souza, Lapouchnian et al. 2013).

Recently, more work has been directed towards inventing more systematic methods for representing and obtaining user feedback and making best use of it during the actual use of software. In (Knauss 2012) the author proposes a process for continuous and context-aware user input that can be used further in community sharing and inform the developers on how to fix problems and debug the system. In (Pagano and Brügge 2013) the authors have conducted an empirical study on the users' involvement for the purpose of software evaluation and evolution and validate a set of hypotheses. In (Ali, Solis et al. 2012) and (Ali, Solis et al. 2011) the crowd feedback was also advocated for shaping software adaptation as users are powerful to capture and communicate certain information that cannot be monitored by automated means and also cannot be fully specified by designers at design time, yet are necessary to plan and enact adaptation.

Moreover, recent research has been focusing on the possibility of utilizing crowdsourcing in requirements engineering (Hosseini, Phalp et al. 2014), (Snijders, Dalpiaz et al. 2014) to cater for the dynamic contexts and diverse users. Crowdsourcing in requirements elicitation has the ability to increase the quality and completeness of requirements elicitation. Crowdsourcing gives the engineers access to a wide diversity of actual and potential users. This would allow engineers, potentially, to gain a wider, and more up-to-date knowledge of how users perceive the system role in meeting their requirements, and to understand how that perception changes over time. Traditional elicitation and user centred approaches, e.g. interviews, focus groups and user-centred design are too expensive to deal efficiently with crowd-oriented applications due to: Limitations in predicting and simulating the actual context of use especially for computing paradigms with inherent high variability and dynamicity of their context such as Mobile Apps and Cloud Services; the unstructured and varied ways in which users provide their feedback typically in a natural language. This lack of rigor requires always a human facilitator to gather, interpret and aggregate what users say. Thus, it would mean some bias and subjectivity. It would also mean a limited scalability

of the evaluation size; capturing the opinion of only an elite group of users which may not be representative to the wider set of actual users. Furthermore, authors in (Sherief, Jiang et al. 2014) stated that the crowd can enrich and keep the timeliness of the engineers' knowledge about software evaluation via their iterative feedback while the software is in use. Although this seems promising, crowdsourcing evaluation introduces a new range of challenges mainly on how to organize the crowd and provide the right platforms to obtain and process their input.

Despite of the speculated benefits of involving users, engineers need to analyse user feedback in order to consider its relevance and possible impact, which involves a number of challenges due to its quantity especially in large systems, quality, structure, and content. Presently, the design and conduct of feedback acquisition are heavily dependent on engineer's creativity. Also, the literature is still limited in providing engineering approaches and foundations to develop systematic approaches for the structuring and use of users' feedback (Almaliki, Ncube et al. 2014, Almaliki, Ncube et al. 2015) and support engineers with appropriate tools.

Also, requirements are gathered from, yet must still represent, a diverse group of users; they are basically volatile in nature. These issues are exacerbated by the problem that users still typically provide their feedback on the fulfilment of their requirements in a natural language and in an ad-hoc manner, which introduces a great deal of imprecision and ambiguity. To cope with such a lack of precision, a range of semi-automated techniques have been suggested to handle such user data. This includes techniques such as text mining and/or human facilitator (Galvis Carreño and Winbladh 2013). These techniques may be used to gather, interpret, aggregate, and revise what users say, partly to mitigate for such issues as bias and subjectivity in their textual responses. More effective results can be reached if the feedback is written in a structured format. Structured feedback text would, arguably, allow approaches, such as text processing, to provide more accurate results within less time and with fewer human interventions. If text is structured the requirements extraction process can be more systematic, eliminating complexity and ambiguity found in natural language, and requiring less effort.

Reviewing the literature, this thesis could not identify systematic approaches for engineering feedback acquisition, communication or requirements updating while the software is in use (i.e. during the maintenance phase). This research focuses on to the development of a new formalized and systematic approach for feedback acquisition and communication during the maintenance phase. This includes devising mechanisms to structure such feedback in a way that makes it easy for users to express and engineers to interpret (Sherief, Abdelmoez et al. 2015). Also, it includes inventing new a new method that allows engineers and end-users to communicate during the maintenance phase using structured feedback. Acquiring and storing the communication threads of interrelated structured feedback would definitely

carry useful information that could enable the extraction of useful information for keeping requirements information up-to-date. Also, it will help in evaluating the overall quality of the system, and would help in performing evolution tasks accurately and making effective decision based on updated information. Moreover, this research suggests benefiting from the large feedback collection from users, by using recommendation and especially social recommender systems that might be useful in pattern mining and decision-making based on user feedbacks. It could be used to determine similarities between users, and accordingly reuse feedbacks.

1.2 Research Aim

In the light of the above mentioned challenges and the lack of engineering approaches for users' feedback acquisition, communication, and requirements extraction and updating of software systems during the maintenance phase, the research aims is twofold that is:

1) To study common feedback structures and their pillars so that acquisition and communication methods which maximize quality without hindering users experience can be provided;
2) Exploring how to extract new requirements to keep requirements models and documentation up-to-date during the maintenance phase. This will lead to a more effective management and richness of the users' role as evaluators. Also, it provides a systematic means for requirements engineers to capture and analyse and prioritize feedbacks and thus requirements too.

1.3 Research Questions

Based on the aim of this thesis the following questions were derived:

1) From the perspective of both end-users and software engineers, what are the engineering foundations and challenges for the empowerment and utilization of users' feedback in software systems?
2) What are the different types of feedback structures and the concepts that constitute each type? What are the rules that govern their usage?
3) How to develop a feedback acquisition and communication method that utilizes feedback to involve end-users as active collaborators and to inform the engineers' maintenance tasks and decisions?
4) How can new requirements knowledge be extracted from the collected end-user feedback to help requirements engineers in keeping requirements information up-to-date?

1.4 Research Objectives

In order to answer the research questions, the following set of objectives has to be met in this research:

Objective 1 - Background Search and Literature Review:

The first objective is to review the relevant work done in the literature in order to analyse what peers have reached regarding the definition of users feedback, how it is analysed, and utilized in different areas. Thus, the gaps can be identified and new methods proposed that can move the research field forward. This objective is an ongoing task of analysis and criticism to the relevant topics, which should be continuously maintained throughout the research (see chapter 2).

Objective 2 - Developing a new classification of feedback components and types:

The second objective encompasses finding a definition for feedback, its types, and identifying and defining the constituent for each type, and the details that users employ to describe their feedback. That is to provide structured feedback that is more meaningful and useful, while the software is in use. (see Chapter 4).

Objective 3 - Designing a novel method for feedback acquisition and communication:

The goal is threefold that is to: 1) provide a systematic means that enables end-users and engineers to utilize feedback structures as a communication tool, and be able to validate the acquired feedback and store it in a structured and interrelated manner; 2) align the process with the software change management and evolution process that engineers utilize during the maintenance phase and that is to embed the feedback acquisition as a core task and driver for the evolution process to inform its tasks and decisions; 3) to integrate the process with feature models that could provide further systematic assistance for the engineers in impact analysis tasks and in extracting problems related to certain features, or determining which features are more problematic (see chapter 5 and 6).

Objective 4 – Extending the feedback acquisition and communication method to enable requirements extraction and updating:

The goal is to help keep the requirements information up-to-date. A novel process will be designed to enforce and automate this task by embedding it in the designed acquisition and communication process achieved from the previous objective, which will help produce efficient and reliable results as they will be extracted from the communication threads that took place between the engineers and end-users (see Chapter 6). Also, it will help in increasing the engineers' enthusiasm towards this tedious yet crucial task.

Objective 5 – Validation of the acceptance of the approach:

In this research, the proposed approach will be verified and validated by applying it in practice with both end-users and engineers to 1) investigate the use of the novel feedback types in practice and whether they are easy-to-use by end-users. 2) Also, to investigate whether they are successful means in providing engineers with useful/ meaningful information that could help them in accomplishing the evolution tasks and taking decisions through the new proposed approach for feedback acquisition and communication. 3) And to design with the engineers how the stored communication threads could inform the requirements extraction and updating method (see Chapter 6).

Objective 6 - Formalizing the feedback structures and the developed methods

The goal is to have a method so simple that it can express any fact, and yet so structured that computer applications can do useful things with it, which in this research case can build a more formal feedback acquisition method that can systematically analyse and classify a given feedback, validate and store feedback in the ontology knowledge base. Thus, an ontology of feedback concepts will be built in order to reach a common definition of the structure of feedback and the rules and relationships that govern its use. That could be used to further query the stored feedback threads to extract requirements information that will update the feature model and specification. Finally, a prototype will be designed and introduced to help visualise how the ontology along with the proposed new approaches could be applied in practice if a real tool existed (see chapter 7).

1.5 Report Structure

This report is structured as follows. Chapter 2 is a background search and literature review, this chapter tries to address relevant areas and topics needed to identify the gaps, and motivate our research. Next, chapter 3 is the explanation of the research methodology that is followed in order to achieve the research aim and objectives. It provides full survey of the different research approaches, research designs and analysis techniques and argues the adopted research approach, research design, and research methods used in this research. Then, chapter 4 explains the two-staged empirical study that was conducted from users' perspective to derive the feedback types and their components. Afterwards, chapter 5 explains the interviews study with the software engineers provides a full explanation of the recognized challenges that helped in deriving the need for the new processes explained in objectives 3 and 4. After that, in chapter 6 the resulted design of the intended outcomes is explained and an explanation of how it was designed and derived from practice with its intended user is described. Then, in chapter 7 the formalization of the proposed approaches is explained. First, a documentation of the Ontology: its purpose, the rules that define it and its validation is explained. Second, the prototypes deigned for engineers are illustrated and

explained. After that, in chapter 8 a discussion, final conclusions and future work are explained. The Structure and Roadmap is illustrated in Figure 1.

1.6 Research Publications

The preliminary results of the initial investigation that aimed at providing results on the different aspects of the feedback design and conduct of feedback acquisition was published at EASE'14 (acceptance rate: 24%).

- Sherief, N., Jiang, N., Hosseini, M., Phalp, K., & Ali, R. (2014, May). Crowdsourcing software evaluation. In Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (p. 19). ACM.
- Sherief, N. (2014, May). Software evaluation via users' feedback at runtime. In Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (p. 58). ACM.

Also, the detailed result of the empirical user study that was performed was published at PoEM 2015 (acceptance rate: 25 %). It discusses the study findings on the structures of users' feedbacks and the contributions it introduces to the literature, and how it can be utilized in the domain of modelling and facilitating user feedback acquisition.

- Sherief, N., Abdelmoez, W., Phalp, K. and Ali, R., 2015, November. Modelling users feedback in crowd-based requirements engineering: An empirical study. In *IFIP Working Conference on The Practice of Enterprise Modelling* (pp. 174-190). Springer International Publishing.
- Sherief, N., 2015, November. Modelling and Facilitating User-Generated Feedback for Enterprise Information Systems Evaluation. In *PoEM* (pp. 117-124).

1.7 Declaration of co-authors contribution to the published work of this thesis

The author of this thesis is the first author of all the resulted publications from this thesis work. The contribution of the first author was as follows:

- Formulating the idea and aim of each paper.
- Deciding upon the research approach and method to be adopted in each paper (e.g. qualitative research approach and methods like: focus groups, interviews).
- Designing and implementing the empirical studies presented in each paper (e.g. developing interview scripts, recruiting the participants, collecting the data…etc.).
- Analysing and interpreting the collected data and draw the conclusions (e.g. qualitative thematic analysis).

- Reporting the findings and fully writing each paper.

The co-authors contributed to the published papers in terms of verifying and validating the studies' findings by comparing them against the actual responses from the participants. They also provided direction and feedback on the structure and the overall articulation and presentation of the papers' message. In addition, they gave remarks on the methodology and also checked the writing quality and suggested modifications on some parts of the text. Furthermore, the co-authors enhanced the papers with the suitable terminologies especially those related to the venue where the papers were published. Also, they suggested literature work to be reviewed and criticised that would help relate the topic to different domains.

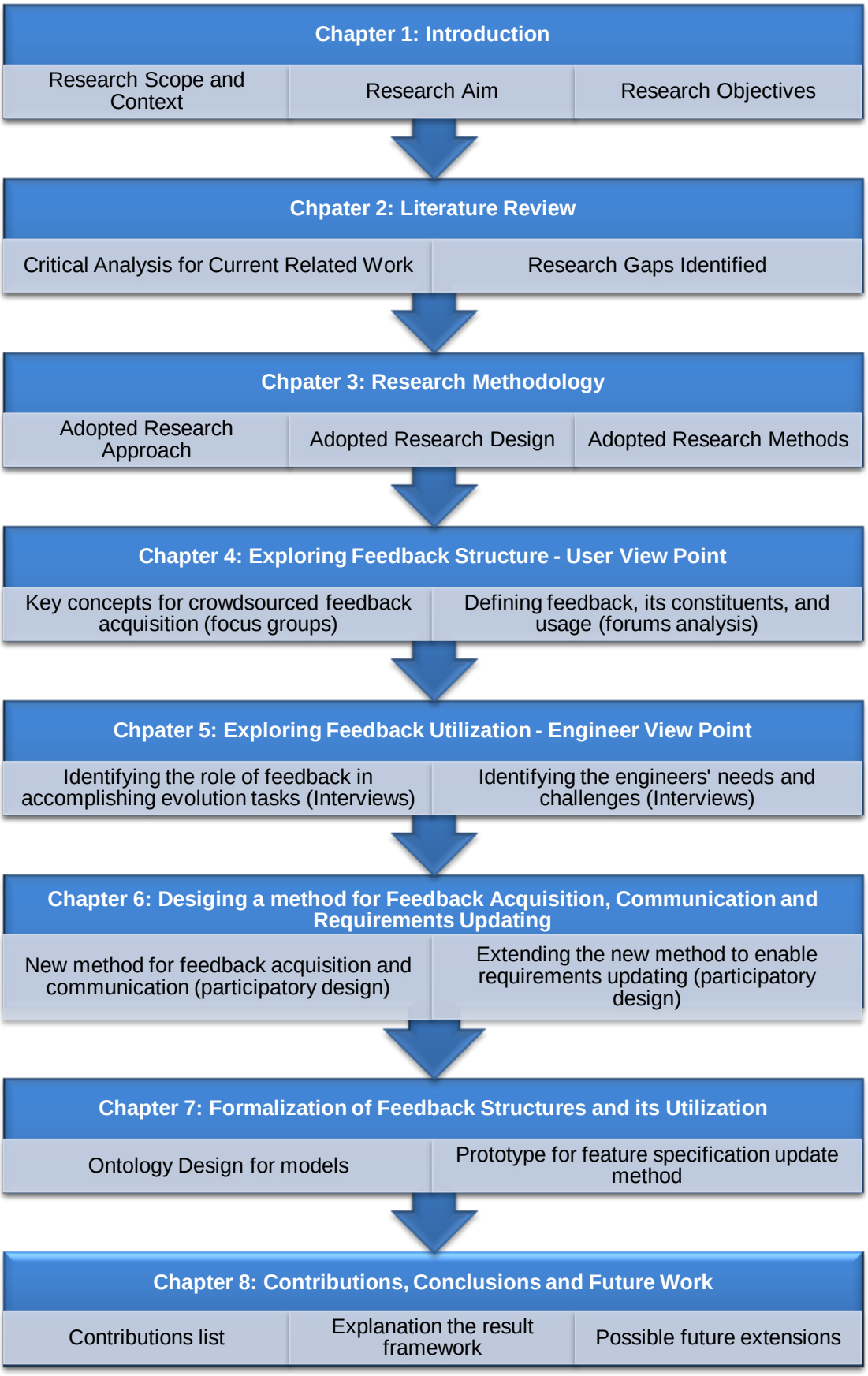

Figure 1. Thesis Chapters and Research Roadmap

1.8 Summary

This chapter gave an introduction to the context and domain of this thesis, discussed the rational for this thesis. It also introduced the aim, research questions, objectives and scope of this thesis. In addition, this chapter gave provided the list of publications that resulted from this research and explained the authors and co-authors contributions in the publications. In the next chapter, a review of the research topics and domains related to this thesis work is presented.

2. Literature Review

In this chapter, a detailed literature review for related topics to this research is discussed. **General topics** regarding how users are involved in traditional approaches and how enterprises benefit from user involvement to communicate problems and enhance their overall process were reviewed. This is covered in Section 2.1 and 2.2. Furthermore, the utilization of the collective end-user feedback was studied, and this is to emphasize its benefits in informing many tasks in the software development. This is covered in section 2.3.

More **Specific Topics** were reviewed from the literature that relates to potential solutions to the problem domain of this thesis. This research intends to develop an engineering approach to systematic feedback acquisition and communication and to develop and engineering method for the semi-automation of requirements extraction and documentation from feedback communication threads that take place between end-users and engineers at the maintenance phase. Thus, this contributes to keeping requirements information up-to-date.

Consequently, there is a crucial need to study and analyse the work that peers have reached regarding the definition of users' feedback, how it is acquired and communicated that is discussed in section 2.4, analysed that is discussed in section 2.5, and utilized that is discussed in a narrowed down scope in section 2.6. Thus, the gaps could be identified and new methods that can move the research field forward could be proposed.

Moreover, fundamental paradigms that provide well established engineering processes for practitioners and researchers regarding the software maintenance and evolution tasks were also discussed to align with and build upon in this research. This is discussed in sections 2.7 and 2.8.

Finally, special purpose paradigms and platforms such as ontologies, controlled natural languages and recommender systems were reviewed with the intension to benefit from them in our proposed objectives, which are discussed in section 2.9, 2.10, and 2.11. Figure 2 below summarizes and illustrated the categorization of the main topics covered in this thesis's literature review.

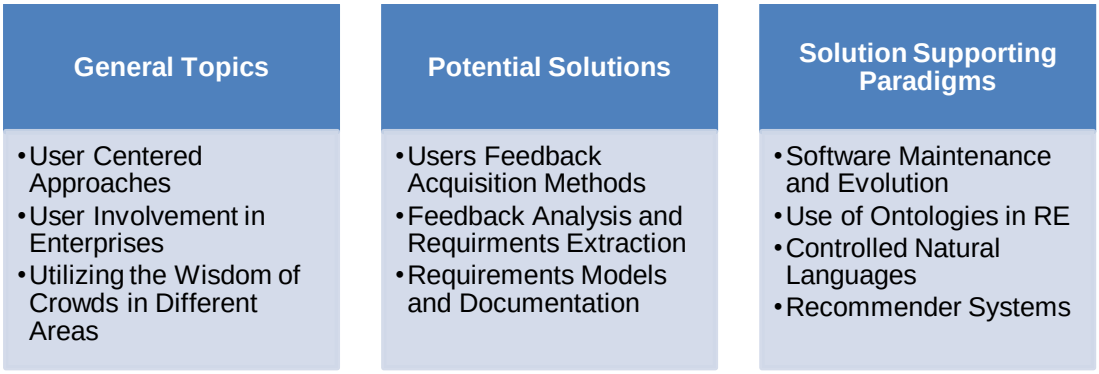

Figure 2. Visual View of the Main Topics Covered in This Thesis's Literature Review

2.1 User Centred Approaches

There are several established approaches where the role of users is central, such as: User centred design (Karel Vredenburg, Ji-Ye Mao et al. 2002), User Experience (Law and Schaik 2010), Agile methodology (Dybå and Dingsøyr 2008), and Usability Testing (Adikari and McDonald 2006). These paradigms were reviewed, as the role of users is central. The purpose was to examine what people in relevant domains do. All these techniques involve users in the software development life cycle, by taking their requirements, and/or involving them in the prototyping and testing. However, these techniques, although they can certainly inform the design of crowdsourced online evaluation and evolution, are expensive and time consuming when used for highly variable software designed to be used by a large crowd in contexts unpredictable at design time.

Furthermore, this work on requirements evolution (Ernst, Borgida et al. 2014) and evaluation from end-users feedback is a kind of end-user computing (Etezadi-Amoli and Farhoomand 1996) in the motivation to provide end users with the ability to change the system according to their views to meet their needs. However, in contrast to end-user computing, crowdsourced evaluation relies on users' feedback as a way to evolve the system, or to adapt the system by switching between configurations at runtime (i.e. while the software is in use) according to the analysis of collective feedback, instead of relying on one user feedback.

2.2 User Involvement in Enterprises

From this research perspective, enabling organizations to effectively communicate their problems and changing/evolving requirements, user empowerment is central. Users act as the primary judgement mechanism in software organizations, and an un-empowered user cannot effectively be a part of creating or contributing to successful enterprise software process improvement. Thus studying user involvement is necessary to identify how users are involved, how their feedback is acquired, and the communication methods they use to

interact with software companies.

Recently a broad study on the user involvement in information systems projects has been published (Eichhorn and Tukel 2016). The authors document and summarize user involvement by identifying studies that investigate user roles and activities, type of communications used, and timing and level of their involvement.

Many user roles and categories have been identified in the research ranging from end-users, user-representatives, to executive management. (Eichhorn and Tukel 2016) argue that capturing a clear list with variety of user roles is necessary for successful user involvement and requirements identification as discussed also in (Jiang, Klein et al. 2009). However, user roles are not yet well understood and carry ambiguity and this affects their proper integration in the development lifecycle.

Also, in the studies reviewed by (Eichhorn and Tukel 2016), mostly users were involved in two main activities: requirements gathering, and quality assurance, while other discuss the importance of involving users as an integral component in more detailed tasks along the entire process of the software development life cycle. Similarly, in this research the involvement of end-users in enterprises was explored to gain more insights about the diverse set of inventive activities they can be involved in.

For example, the implementation of business intelligence (BI) systems is a complex task requiring extensive efforts and resources to create and maintain them. (Yeoh and Koronios 2010), discuss the key success factors of implementing such systems. Their main contribution is the theoretical understanding of user involvement as a key success factor in implementing BI solutions, where they can continuously evaluate the information through their feedbacks and therefore the system will be changed, optimized and developed according to their input. (Barone, Yu et al. 2010) discuss a Business Intelligence Model that aims to enable business users to extract business operations, strategies, and performance indicators in a way that can be linked to enterprise data to provide them with query and reasoning facilities. (Pourshahid, Amyot et al. 2009) suggest users involvement in developing Business Process Management projects using an integrated and tool-supported methodology to help users who are modelling business processes and validating them. Their modelling approach involves using User Requirements notation that integrates goals and usage scenarios, from which requirements can evolve. (Svee, Giannoulis et al. 2011) explore whether and how consumer values impact business strategy and thus how this impact echoes on the solutions that are developed to support business strategy execution. They present how strategy maps can be augmented by consumer values. They link strategy map goals to Holbrook's consumer value typology then they extend the strategy maps and balanced scorecards meta-model to include goals reflecting consumer values, which can be used further as high level requirements for new solutions.

Moreover, (Eichhorn and Tukel 2016) discuss that communication is critical in large software projects, were formal and informal methods are used to ensure project success. Agile methodology improves information sharing and communication as discussed by (Hoda, Noble et al. 2011). However, agile teams rely on gathering requirements information continuously from user representatives in the form of user stories that could be converted to actionable executable scenarios. This enhances the teams' understanding about the customer's needs and ensures that the iterations are developed in a timely manner and straight to the point, which is the main focus of agile methodologies. However, in case of the lack of proper technical documentation, there will be difficulty in maintaining the software on the long run, which could be time-consuming and requirements can easily grow to an unmanageable size, causing inconsistency among information.

All the current work contributes to enriching the understanding of the importance of the role of users in driving the enterprise business process and the different aspects that need to be considered for running these processes. However, the reviewed work operates on the management of requirements at a high level, to ensure goal satisfaction, and business strategy implementation. Yet, there is still a need to develop more systematic methods for feedback acquisition and communication that can manage effectively the collective continuous users' feedback to extract all possible kinds of meaningful information such as: problems, awareness, or new/changing requirements, while providing flexibility and openness for users to continuously express themselves while the software is in use rather than just involving them at the early or final stages. Also, providing a means for structured input of feedback for both end-users and engineers would enable the systematic analysis of information that could lead to an automated means for extracting useful requirements information and keeping it up-to-date, which is an important aspect for a more efficient and accurate maintenance of the software.

2.3 Utilizing the Wisdom of the Crowd in Different Areas

Crowdsourcing is an emerging online paradigm for problem solving which involves a large number of people often recruited on a voluntary basis. It harnesses the power of the crowd for minimizing costs and, also, to solve problems which inherently require a large, decentralized and diverse crowd.

The wisdom of the crowd supports the idea that decisions made collectively by a diverse crowd could be better than the decisions made by a selective group of people who are not necessarily representative enough (Surowiecki 2005).

"Crowdsourcing is an emerging business model where tasks are accomplished by the general public" (Hosseini, Phalp et al. 2014). Crowdsourcing influences across all social and business communications. It is changing the way businesses work, hire, research, make

and market. A Famous Example is Wikipedia. Instead of Wikipedia creating an encyclopaedia on their own, hiring writers and editors, they gave a crowd the ability to create the information on their own (Kittur and Kraut 2008).

Crowdsourcing's biggest benefit is the ability to receive better quality results, since several people offer their best ideas, skills, & support, as opposed to receiving the best entry from a single provider (Brabham 2008). Results can be delivered much quicker than conventional methods. Still, clear instructions are essential in crowdsourcing. Quality can be difficult to evaluate if accurate expectations are not clearly stated.

The essence of crowdsourcing lies in empowering people by giving them a greater voice to take more active role and collaborate in different tasks, which is the same drive for this research. This thesis reviews several work done in crowdsourcing as we share common interest in empowering end-users to take a more active role, which in this thesis is relying on their collective judgement during the maintenance phase to provide feedback about how the software meets their needs and expectations.

In this thesis, the potential of crowdsourcing for software evaluation is advocated. This is especially true in the case of complex and highly variable software systems, which work in diverse, even unpredictable, contexts. The crowd can enrich and keep the timeliness of the developers' knowledge about software evaluation via their iterative feedback.

In this section the work published in the literature regarding crowdsourcing as a method for collecting participants' (mostly end-users) feedback and utilizing it in different tasks is analysed. The analysis of work in this section will range from general research that addresses the possible different uses of crowdsourcing in enterprises, to the use of crowdsourcing in specific tasks such as: conducting empirical studies, software evaluation, requirements engineering problems, and finally its use in the latest emerging research areas.

2.3.1 Practices of Crowdsourcing within Enterprises

First some of the work on Microblogging is reviewed, which is known as the act of providing short messages (possibly feedback) to a website. For example, Twitter is considered a microblogging site. As discussed in this section, microblogs offer a unique source of information gathered from collective users' input on a topic. This could be further analysed to evaluate software systems or share expertise. Similarly, Crowdsourcing is used to harness knowledge and skills of a group of people to solve a problem or contribute content. So both topics are connected like: relying on the wisdom of the Twitter (microblog) crowds as suggested by (Ghosh, Sharma et al. 2012).

The continuously growing technological developments in social networking platforms offer

capabilities to update information in real–time (e.g. in Facebook, and twitter). This has allowed a user's online presence to be transient and dynamic in nature. In this context, micro-blogging has been widely employed by users as a useful means to capture and circulate their thoughts and actions to a larger audience on a daily basis. Microblogs offer a unique information source to analyse and understand context in real-time – i.e. benefits, plans, and activities. The reason why this area is explored is to emphasise on the importance of collective users' feedback as a means to evaluate software systems to meet their needs and expectations, and generate value to enterprises.

(Huh, Jones et al. 2007) explored the use of micro-blogs in a business community. They provide a preliminary investigation, in which they have interviewed a number of bloggers to investigate the effectiveness of blogging in communicating the user's opinions, expertise and questions within an enterprise. Meanwhile, the results emphasize the need to limit corporate users' disclosure to information input and indicate the growing importance of personal brand building and privacy issues in today's enterprises (Schöndienst, Krasnova et al. 2011).

(Banerjee, Chakraborty et al. 2009) gather data from the free timeline of Twitter crossing ten world-wide cities. They worked on this dataset to: 1) explore how users express interests in real–time through micro–blogs; and 2) understand how text mining techniques can be applied to interpret real-time context of a user based tweets.

Moreover, (Bougie, Starke et al. 2011) explore the possible use of micro-blogging by software engineers. They used qualitative analysis approach to analyse threads on Twitter and have found demonstrations of intelligible conversations taking place specifically on Software Engineering topics. These cases included discussion of the current tasks developers are working on, and in several cases, attempts to find solutions to related issues they meet, Self-promotion, Complaints, and the Use of a Specific Tool for Work.

A recent study conducted by (Hosseini, Moore et al. 2015) to investigate the current use of crowdsourcing in the practice of modern enterprises. The study discussed the current practices of the WoC in 33 different UK enterprises by involving more than 60 senior management participants. The study captured that WoC is applied in cases of:

- "When there is a lack of knowledge on certain subjects": specifically in the concept phase WoC could be used to help set the goals and objectives for a project where there is a lack of knowledge at the managerial level or within the enterprise.
- When people in the enterprise are too involved in their business sometimes they fail to analyse their own domain, as they "take many details for granted" and focus on major issues. So using WoC could help providing a fresh external perspective.
- "When deciding on future development": the WoC can be utilized to provide their diverse visions which could help the enterprise in strategic planning decisions.

- "When constructive criticism is needed", where managers stated that external opinions could help give balance to their designs, as they can neutrally highlight the weak points that may require improvements.
- "When feedback is needed to improve quality": feedback from interested and involved clients or ex-clients was considered important as it can help in enhancement procedures and sustain quality.

Besides these benefits, the research discussed that managers also mentioned two main drawbacks, which are: 1) it might allow for untrustworthy participants to take place in the WoC process which might mislead or fail the business activities affecting the quality of the product or service. This was also discussed in (Kittur and Kraut 2008); 2) the might be some cultural issues among the crowd which might make their utilization less effective. Thus, less valid outputs could be retrieved, as it does not come from a diverse crowd with different skills, knowledge and backgrounds.

2.3.2 Crowdsourcing for Empirical Studies

In general, when designing an empirical study in software engineering, engaging the necessary type of participants and appropriate number is always a challenge. Most of the time, researchers are required to perform trade-offs to be able to perform the study (Kittur, Chi et al. 2008). Otherwise, it will take much more time to select the most useful participants, or trying to reach a high number of diverse participants to be able to generalize results. As an alternative to such type of studies, the authors in (Stolee and Elbaum 2010) suggest the use of crowdsourcing to address such a challenge. Moreover, they use Amazon's mechanical Turk as a tool that allows them to easily create, manage crowd-sourced studies, perform prerequisite qualification tests for filtering participants, ensure privacy, manage payments, and collect results. Yet Mechanical Turk has its learning overheads, as the researcher should be aware of its capabilities and how to use the tool and understand the underlying technologies such as XML, web services, and shell scripting. These studies relate to our research in two aspects. First, it provides an alternative way of performing empirical studies, which is part of our research objectives for developing a framework for feedback acquisition and utilization. Second, it emphasises the role of end-users in the feedback providing for evaluation process. Yet our research addresses the user feedback formal structuring for deployed software evaluation, and developing systematic methods for benefiting from crowd-sourced feedbacks which adds new scope and usage even beyond what these papers propose.

Several researches have been held to use crowd-sourcing in the area of software usability testing or interface evaluations, as it serves as an alternative approach to lab-experiments that are expensive and time-consuming. For example, (Liu, Bias et al. 2012) used crowd-sourcing in evaluating the usability of graduate school's website. They used Mechanical

Turk as a platform for performing their tests, which easily helped them manage the test. They discuss several advantages and disadvantages for crowd-sourced usability testing have over a similar lab usability test. The advantages are: more participants' involvement, low cost, high speed, and various users' backgrounds. While the disadvantages include: lower quality feedback, less interactions, more spammers, less focused user groups. Therefore, this research emphasizes that crowdsourcing could be a very good option in software usability testing specially with development teams with short time and low budgets. However it should be designed and used carefully as it imposes several challenges.

In another similar research, (Komarov, Reinecke et al. 2013) also suggest the use of Amazon's Mechanical Turk as a crowd-sourcing platform to evaluate the performance of user interfaces. This study did not show any significant differences between lab evaluations and crowd-sourcing. The following measures were used: the mean task completion times, consistency of participant performance through the experiment, error rates, utilization as the fraction of times when the user used the novel interaction mechanisms when one was available, fraction of participants who were classified as extreme outliers. In this study no significant differences between Turkers and lab participants was detected in any of the measures. Yet, the use of Mechanical Turk platform for crowdsourcing showed better results, especially when the main measures of interest are the task completion time, and error rates.

Nevertheless, the above researches treat the crowdsourcing as a one unit, which is the collection of feedback, without addressing its peculiarities. For example, the formal analyses and tailoring of crowd-sourced feedback into the process of software testing and evaluation process has not yet been addressed. It is also an ideal pool for collecting user experiences that could be shared and collectively analysed to enhance future user experience.

This novel approach definitively shows lots of potential, but still a more thorough method should be developed to address the different dimensions and challenges of such an approach in its different configurations.

2.3.3 Crowdsourcing for Software Evaluation

Crowdsourcing harnesses the power of the crowd for minimizing costs and, also, to solve problems which inherently require a large, decentralized and diverse crowd. In this paper (Sherief, Jiang et al. 2014), the researchers advocate the potential of crowdsourcing for software evaluation. This is especially true in the case of complex and highly variable software systems, which work in diverse, even unpredictable, contexts.

Although this seems promising, crowdsourcing evaluation introduces a new range of challenges mainly on how to organize the crowd and provide the right platforms to obtain and process their input. This paper, which is part of this thesis work focuses on the activity

of acquiring evaluation feedback from the crowd, and utilizing it to inform the evolution tasks that engineers perform, and also to help keep the requirements knowledge up to date during the maintenance phase via their iterative feedback.

This paper proposed a systematic development of a crowdsourcing-based solution to software evaluation. While the concept of crowdsourcing is shown to be promising considering the increasing complexity and diversity of contexts for current systems, there is still a lack of foundations on how to engineer it and ensure correctness and maximize quality. This paper focused on the activity of interacting with users and getting their feedback on software quality as one important step for a holistic approach for crowdsourced software evaluation.

2.3.4 Crowdsourcing for Requirements Engineering

The software requirements are description of features and functionalities of the target system. Requirements convey the stakeholders' needs and expectations for the software product or service. The process to gather the software requirements from client, analyse and document them is known as requirement engineering. The goal of requirement engineering (RE) is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document (Kotonya and Sommerville 1998).

Requirements Elicitation is the process of discovering, reviewing, documenting, and understanding the user's needs and constraints for an intended software system by communicating with customer, end-users, management users and others who are the key stakeholders in the software system development (Kotonya and Sommerville 1998).

The longer the system's lifespan, the further it is subject to changes in the requirements that result from changes in the needs, concerns and expectations of its stakeholders. Crowdsourcing can support requirements elicitation, especially for systems used by a variety of users and operating in a dynamic context and changing platforms where requirements frequently evolve.

(Hosseini, Shahri et al. 2015) follow an empirical approach to study how to support the use of crowdsourcing for requirements elicitation. Their work intended to investigate ways to configure crowdsourcing to improve the quality of elicited requirements. Despite the potential of the idea and the support they gathered from participants and experts, there is no much literature in this area. Yet, their work introduced and discussed a set of risks accompanied by adopting such approach, which makes the decision with adopting it and configuring it in the correct way challenging. This is to help researchers and practitioners adopt the idea and move the field forward.

Similarly, (Wang, Wang et al. 2014) used crowdsourcing to acquire requirements, but with

attention given to the problem of employing participants with specific narrow down domain knowledge. They proposed a recruitment framework for software requirements acquisition based on the Spatio-temporal availability of participants. Their theoretical analysis and simulation experiments showed the feasibility of their proposed framework.

Further advances in this area of research recently came to applying the gamification concept known as “the use of game design elements in non-gaming contexts” (Deterding, Sicart et al. 2011) to crowdsourcing in order to inform and enrich the requirements engineering discipline. Initially, (Snijders, Dalpiaz et al. 2015) introduced a requirements elicitation tool named *REfine* that is a gamified online platform for requirements elicitation. A case study showed its potential of the approach for improving RE in software production. Gamification was introduced as a chance to increase the engagement of stakeholders in requirements engineering by creating feedback loops that motivates the valuable participation by rewarding effective participants, i.e., those that provide valued inputs (requirements) for the system under design. Then, the related trend of crowdsourcing was also employed by (Dalpiaz, Snijders et al. 2017) to support the broad and diverse participation of stakeholders, by making RE a participatory activity where current and potential users, developers, customers and analysts are involved.

2.3.5 Crowdsourcing for Software Evolution

Software evolution and maintenance are among the earliest areas that have benefited from crowdsourcing. Traditional formal or automated verification methods may fail to support large software systems. To help overcome this issue, a market-based software evolution method was proposed by (Bacon, Chen et al. 2009). The goal of the method is not to ensure the total ‘correctness’ of software system, but to inexpensively resolve bugs that users care about most. The proposed mechanism lets users prioritize bug fixes, new enhancements, or new features, and incentivizes the responders.

Other authors attempted to enrich the idea by performing empirical studies to capture the different aspects of user involvement. In (Pagano and Brügge 2013), authors have conducted an interesting empirical study on the user involvement for the purpose of software evaluation and evolution. They came up with several hypotheses that contribute to three different aspects of user involvement, which are: user involvement settings, user involvement workflow, and user involvement requirements for tool support. Their study did not pay much attention to the earlier stage where feedback acquisition activity takes, how the collected feedback could be utilized as a communication means with developers, nor how the feedback could inform the evolution tasks.

In a similar empirical study (Pagano and Maalej 2013) on user feedback in the AppStore, the authors declare that the AppStores serve as a very interesting media for feedback to both developers and end-users. Their empirical study provides results to the three aspects

they were investigating, which are: how and when users provide feedback, the feedback content, and the impact of feedback on user community. These researches are still unsystematic as they do not model feedback in a formal way to be understandable and usable by both users and developers. Also they do not address methodical ways to analyse, process and benefit from the collective judgements of users in software evaluation.

Furthermore, in the last decade there has been a lot of interest in the area of engineering runtime self-adaptive systems (Salehie and Tahvildari 2009). By runtime it is meant while the software is in use and therefore means while the software is evolving during the maintenance phase. In spite of its importance, the role of users in supporting and tailoring the adaptation process and decisions is still unclearly presented. The involvement of users as partners with the adaptation process amplifies its potential and range of applications (Cámara, Moreno et al. 2015) (Mistrik, Ali et al. 2016). It is more powerful for crowd-sourcing users to act as collaborators and monitors for adaptive systems. (Ali, Solis et al. 2011) argue that users' perception is powerful to capture and communicate certain information that cannot be captured by automated means and are necessary to plan and enact adaptation.

The ultimate goal of adaptation is to maintain and improve the role of software in meeting users' evolving requirements (Ali, Solis et al. 2012). This motivates scholars to develop approaches for user requirements model-driven feedback structuring, as models can present the prominent aspects of adaptation from user perspective, and when formalized they enable the automated reasoning of software adaptation. Nevertheless it opens the door to several research challenges.

Further work has been carried to identify main advantages, domain areas and the challenges triggered by the mechanisms for acquiring user feedback to guide the adaptation process (Almaliki, Faniyi et al. 2014). The authors conducted a two-phase expert survey on the topic of Social Adaptation. Their results have concluded that there is a lack of models and mechanisms for supporting this concept. Also, engineering approaches are highly needed for Social Adaptation to empower collaborative users' involvement in shaping adaptation decisions and to systematically develop the feedback acquisition process and interaction styles.

Additional work was done on for crowdsourced adaptation. For example, (Akiki, Bandara et al. 2013) focused on utilising crowdsourcing for user interface adaptations. Their motivation is based upon that there are complex software systems like enterprise systems that contain many features which increases the visual complexity of the software. Since the end-users only use a distinct small subset of the system features, the authors proposed an approach based on model-driven user interface construction which enables the crowd to adapt the interfaces via an online editing tool. A preliminary online user study pointed to promising findings for usability, efficiency and effectiveness.

2.4 Users' Feedback Acquisition Methods

Users' feedback acquisition methods (if they exist at all) in software systems are not systematic and rather ad hoc. No much literature is known about how the way software engineers collect and work with users' feedback.

(Maalej and Pagano 2011) propose a social engineering process that enable software engineering teams to develop and maintain social software systems via continuous end-user feedback. They combine activities that are already performed in existing engineering processes (such as usability optimization, or prioritizing feature requests), yet in an isolated way. However, their work lacks the ability to integrate directly into the software applications, and also lacks an integrated systematic means for communication channels from the software engineers to the end-users. The existence of such a channel can always keep end-users aware of the actions and changes their given feedback caused on the system. This can highly improve their satisfaction and trust in the software.

Apart from error reports, there is no commonly agreed practice on how to provide or gather user feedback during software evolution. (Pagano and Brügge 2013) conducted an empirical case study on five software development companies to explore the current practice of users' involvement via their feedback. Their study mainly focused on the stages after feedback has been collected, and no much emphasis was made on the initial stage where the feedback acquisition activity happens. Thus, there is a high need for tools and approaches to support the process of collecting, structuring end-users' feedback.

More work has been directed towards inventing more systematic methods for representing and obtaining user feedback and making best use of it. In (Maalej, Happel et al. 2009), the authors have made an attempt to introduce such an idea. They introduced a process for continuous and context-aware user input that can be used further in community sharing, and intelligently enhancing the system through collective judgements. However, they deal with feedback from a very general perspective without looking into details that regard its quality. Also, a systematic practice to gather users' feedback is still missing in their work, which affects the utilization of their methods to obtain useful and meaningful information that could inform the software evaluation, and maintenance. Also in (Knauss 2012) the author investigates existing requirements elicitation techniques, and suggests that the collective judgements of stakeholders is beneficial to requirements elicitation and discusses their potential for considering context.

A more recent study conducted by (Almaliki, Ncube et al. 2014) reported an empirical mixed method study to explore and investigate users' behaviour with regard to feedback acquisition in software applications. Their results show that there is a need for systematic approaches for supporting adaptive feedback acquisition that should fit and adapt to each

different user type, and should highly consider the factors that influence users' behaviour during the feedback acquisition process.

In a further study (Almaliki, Ncube et al. 2015) provide a clearer view and a deeper comprehention of users' different behaviours to feedback acquisition by demonstrating seven personas of users' behaviour to feedback acquisition. Again, this emphasizes the need for an adaptive feedback acquisition to assist these various behaviours. Additionally, their work gives a clear understanding to software engineers when devising an adaptive feedback acquisition. This work was further extended (Almaliki and Ali 2016) to study the cultural differences among users that also play a role in affecting their stimuluses to feedback acquisition. The paper also supports the need to have a persuasive and culture-aware feedback acquisition which opens the gate for further research in this area.

It could be concluded that the literature is still limited in providing engineering approaches to developing systematic feedback acquisition and communication. This research focuses on to the development of feedback modelling and elicitation framework. This includes devising mechanisms to structure such feedback in a way that makes it easy for users to express and engineers to interpret. This will allow the system to prioritize different problems reported by users. Also, it will help in evaluating the overall quality of the system and in taking evolution and maintenance decisions.

2.5 Feedback Analysis and Requirements Extraction

In this section the efforts made to extract requirements information from users' feedback is introduced. Various works has been done on how to elicit requirements from users' feedbacks for the purpose of supporting software evolution. The main goal is to elicit new/changed requirements from large sets of users' feedback. The problem in extracting such information is the large volume of data that has to be analysed, the time commitment required to perform such task and the considerable human interventions that conveys a great deal of bias/subjectivity. Therefore, researchers have been aiming at trying to use new methods in order to alleviate part of the process.

Authors in (Galvis Carreño and Winbladh 2013) use the topic extraction mechanism to process users' comments. Their process includes tokenizing and removing noise from input data (i.e. users' feedbacks). Then, they extract the main topics mentioned in the feedback, along with some sentences demonstrative to those topics using sentiment analysis. This information has to be revised by requirements engineers who use it to plan for next software versions. However, a main drawback is that using only sentiment analysis and topic modelling does not provide associations between topics and attitudes which are crucial for informing requirements engineers about requirements changes. The main goal of this work was to automatically generate a report for requirements engineers with the list of

new/changed requirements. So they wanted to test if their approach saves time, is it possible to extract the same information in less time and with less effort, and whether it is better to use the generated report compared to the original list of manually extracted comments.

Also in (Schneider 2011), authors propose deriving change requests and new requirements from spontaneous feedback gained in real usage settings. To do so they have defined a simple domain ontology consisting of generic broad types of feedbacks and associations. They used clustering techniques to cluster feedback messages according to the entities they refer to. Then they proposed applying existing formal techniques to feedback texts for in-depth interpretations. They suggested using natural language parsing where stop words are removed and keywords are searched, and heuristic filtering that can match the detected keywords to domain ontology.

In (Seyff, Graf et al. 2010), the main focus of the research was to develop a mobile tool to capture the users feedback whenever and wherever they want. The tool also gathers contextual information to enrich requirements descriptions and to provide information about the end-users environments. Finally, the tool captures the importance of the task to the user. This research aims on providing an elicitation approach that can offer new opportunities for users to support them in documenting their needs. Authors did not explore how to support requirements engineers in analysing and transcribing end-users needs into well-defined requirements.

In (Pagano 2011), the author proposes a framework for systematic analysis of continuous user input, and the enabling techniques that can be used to support the process. The techniques proposed in the framework are all existing techniques. First, the author suggests analysing the gathered user input using text mining and information retrieval to identify domain concepts. Then, these domain concepts are used to identify fundamental structure of user input using clustering techniques and tagging. Afterwards, the data can be filtered and prioritized to be presented to the requirements engineers with reduced amount of redundant information using social network analysis and collaborative filtering techniques.

Other researchers are using mining of feedbacks, but for different areas other than software evolution. The increasing ubiquity of the Internet has radically changed the way that consumers shop for products. Consumer-generated product reviews (i.e. feedbacks) have become a useful source of information for customers, who read the reviews and decide whether to buy the product based on the information provided.

For example, in (Dave, Lawrence et al. 2003, Hu and Liu 2004, Liu, Hu et al. 2005), authors are using several mining techniques for mining customer products reviews. It is a common way that merchants selling products on the Web ask their customers to evaluate the products and related services. This makes it difficult for a prospective customer to read

them in order to make a decision on whether to buy the product.

In (Hu and Liu 2004) authors aim to summarize all the customer reviews of a product. They are only interested in the specific features of the product that customers have opinions on and also whether the opinions are positive or negative (same aim in (Dave, Lawrence et al. 2003)). A number of techniques are presented to mine such features. The system first downloads (or crawls) all the reviews, and puts them in the review database. The feature extraction function, which is the focus of this paper, first extracts “hot” features that a lot of people have expressed their opinions on in their reviews, and then finds those infrequent ones. In order to do this, they use association rule mining to find all frequent item sets. They also use Part-of-speech tagging (Authors in (Liu, Hu et al. 2005) also use POS tagging), is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition as well as its context, and is used as a pre-processing step before applying association mining algorithm to discover frequent nouns and noun phrases.

Similarly, in (Archak, Ghose et al. 2011), the authors use techniques that decompose the reviews into segments that evaluate the individual characteristics of a product. Towards this goal, they develop a novel hybrid technique combining text mining and econometrics that models consumer product reviews as elements in a tensor product of feature and evaluation spaces.

All these attempts to extract requirements from users’ feedback, are using existing techniques to try to find meaningful information (i.e. requirements) hidden inside users’ texts. Since the text is written in natural language (i.e. unstructured) therefore it can carry different meanings and interpretations. This lack of rigor requires always semi-automated handling of data (i.e. includes techniques of text mining and/or human facilitator) to gather, interpret, aggregate, and revise what users say, which may contain some bias and subjectivity.

These efforts would have led to more effective results, if the feedback was written in a structured format (Sherief, Abdelmoez et al. 2015). A structured feedback text would allow such approaches using text processing techniques to provide more accurate results that require less time and human interventions. If text is structured using syntax and semantics, the requirements extraction process can be more systematic, eliminating complexity and ambiguity found in natural language, and requiring remarkably less effort.

It is also worth mentioning that this research focuses on written information as this is the main method for feedback acquisition and communication. However, there are other methods for communication such as surveys, which is considered to be a restrictive feedback tool, as it relies on a set of predefined questions with direct specific answers and little space for end-users to elaborate and express themselves. Furthermore, software

engineers could gather requirements from end-users using voice methods (whether phone calls, skype calls…). However, in order to properly utilize this method as a means for requirements extraction and documentation, this content would need to be transcribed into textual format to deal with, as it is unlikely to use audio records in software documentations.

2.6 Requirements Models and Requirements Documentation

This research also gives a particular focus on studying the requirements engineering models which support variability. Models can represent the prominent aspects of the software that when formally used enables automated reasoning to derive essential information from the software employing them. Since this research is proposing a new crowdsourced evaluation process, using these models to represent stakeholder's goals, software features, configurations, and relating users' feedback to them would be easy to the users. Also, this will provide systematic assistance to the engineers in interpreting and extracting new requirements and problems.

2.6.1 Requirements Models and its Utilization

One mainstream technique is goal modelling (Yu 2009). Goal models fit the early stages of the software development and explain the functionality a system to operate and why to operate it. Goal models are very useful in specifying both functional and non-functional requirements. Functional requirements are complete if they can be all mapped to goals, and all goals are satisfied. Non-functional requirements can be specified as soft goals. Goal models can be used to represent the impact of different solution approaches on soft goal satisfaction. Goal models participate as a very important candidate model in our intended engineering framework. This is because goal models help in clarifying requirements and linking them to correct goals without missing any requirements. More importantly, they enable requirements' completeness to be measured, as requirements can be considered complete if they fulfil all the goals in the goal model. Therefore, after users provide feedback about the software features this could be linked to the system requirements to inform better evaluation of the main system goals. This hierarchy will also help analysts and developers to trace and plan how updated or new requirements will fit into the main system model.

Furthermore, a feature is defined as a "prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems" (Kang, Kim et al. 1998). The core of a feature model is a feature graph. A feature graph represents variability in a very compact and clear way, in that it presents the features in a tree of AND/OR nodes to identify the common and variable parts within the domain. Feature Model is naturally understood by users as it talks to them in their own terms, i.e. what they see in a system.

This makes it a good candidate for using it in the intended framework in order to help

customers in linking the feedback to a specific feature, which will help in: accurately specifying the modification scope of the reported problem, or requested enhancement or feature; also this would help engineers in performing the impact analysis tasks, which is one of the most important tasks in the change management process that hinders the successful change implementation; Furthermore, by introducing changes to the specified feature, this would also help in keeping the requirements information up-to-date after the change is accepted and closed, which would yield more accurate results along the maintenance phase of the software.

Goal models can be utilized to represent the stakeholders' goals. Also, it can be related to the feature model where together they represent both the functional and non-functional requirements of the system. By relating both models to the structured feedbacks, engineers can propagate through the interconnections between them to determine different levels of evaluation information. For example, by looking on the feedbacks and their related features in the feature model they can identify most problematic features in the software according to some simple metrics like the number of negative feedbacks referencing that feature. Or they can look at it from a higher level to see which goals are violated keeping stakeholders unsatisfied.

Business process modelling (Group 2006), often called process modelling is the analytical representation or illustration of an organization's business processes. It is widely viewed as a critical component in successful business process management (BPM). It is used to map out an organization's current processes to create a baseline for process improvements and to design future processes with those improvements incorporated. Process modelling often uses Business Process Modelling Notation (BPMN), a standard method of illustrating processes with flowchart-like diagrams that can be easily understood by both customers and engineers. Even though Business Process Modelling relates to many aspects of management (business, organisation, profit, projects, etc) its detailed technical nature and process-emphasis link it closely with change management programs that are involved to put any improved business processes into practice.

In the intended framework of this research, using both feature and business process models to represent the small-grained features of the system is considered essential. The argument is that linking features with the business process would provide further benefits. As by identifying the affected business process this would improve the analysis task in the evolution process, because it would provide information about how the business process will be affected by the change and the impact of the change on the larger scale. Moreover, this research claims that it would improve the customers-to-engineers and engineer-to-engineer communication by informing all the affected parties by the introduced changes. Furthermore, linking feedback to both business process and specific feature would help in the similarity assessments when recommending possible actions from previous problems.

These advantages improve the realization of objective 3, which aims at developing a new method for feedback acquisition and communication that helps in informing the evolution process tasks. Using feedback threads that are linked to requirements models in the communication will help engineers in the accurate scope identification and impact analysis during the evolution process. Also, linking feedbacks to feature models would inform the realization of objective 4, because it would facilitate accurate requirements extraction, and documentation updating. Furthermore, linking feedback to business process models would add a new perspective in the documentation, which is how new requirements or changes are linked to customer's goals or how they could affect them.

2.6.2 Requirements Documentation Challenges

A Requirements Document (RD) is a formal contract between the software company and the customer for a product. It describes in full detail all the features, and processes that should be implemented. A RD is used through the entire cycle of the project to ensure that the product meets the detailed specifications and that the project achieves the desired results. However, keeping this documentation up-to-date especially during the maintenance phase where changes occur regularly is still a challenging task.

The main task in requirements documentation is information collection. In the early stages of the software development life cycle this is done through brainstorming and interviews with various sources, including developers, customers, engineers and end-users. The collected information should be documented in a clear and concise way, familiar to the business user, to ensure successful product development and high-quality end-product. However, the same task is not done in a clear systematic way during the late phases specifically during the maintenance phase. This issue is due to the ad-hoc manner the communication is handled with customers in this stage, which leads to lost information and lack of consistency.

In a study done on 18 different organizations, (Kajko-Mattsson 2005) confirmed several documentation problems during the maintenance phase. Examples are: Software systems are not continuously documented at all granularity levels for example requirements documentation need to carry information about features, use cases, goals, and technical information; The majority of the organizations do not provide guidelines for how to document their software systems, which is left for the engineer's perspective, knowledge, experience and skills to document what he think is necessary; Also, support for making decisions about future changes is poor. Maintainers have vague insight into all the corrective modifications made to the system and their history. Hence, they cannot effectively evaluate the quality of their systems and the effectiveness of their development and maintenance methodologies.

Several further researches were reviewed, that identified that the key maintenance problem is the lack of up-to-date documentation (de Souza, Anquetil et al. 2005). Their purpose was

to examine what types of information documentations are needed by software maintainers that have already been considered in other studies. Examples were: hierarchical architectures of the system, step-by-step instructions for users, requirements description and design specifications for experts, business rules, and histories, tests, data model, class model, business process description, user manual, and project minutes for XP projects.

(Forward and Lethbridge 2002) reached the same results by conducting a survey to capture the documentations needs, usage, and other attributes. They conclude that documentation is an important tool for communication and should always serve a purpose.

Furthermore, (Anquetil, Oliveira et al. 2005) present a re-documentation tool to partially automate the documentation process during the maintenance phase. However, their research treats the documentation as a reverse engineering process. This research focuses on the documentation of functional (i.e. feature) specifications, and the updating of feature models and their interdependencies. This is to provide full documentation of how the system works in a way that could be understandable by end-users and beneficial for software engineers, as it will be extracted from their feedback communication threads. Also, this research suggests catering for the documentation task as an embedded sub-process within the maintenance phase.

Finally, a more recent experiment was conducted by (Leotta, Ricca et al. 2013) on 21 bachelor student to highlight the importance and impact of accurate and up-to-date documentation on maintenance and to identify the challenges that the engineers encounter and hinders their ability and enthusiasm in documenting the changes accurately.

In this research the challenges discussed are main concerns that act as a driver for developing a new embedded process that provide systematic means for extracting requirements information during the maintenance phase and updating the requirements documentation and models during the maintenance phase. As a starting point this could be managed by linking every feedback provided by the end-user or software engineer to a specific feature and process in the software system.

2.7 Software Maintenance and Evolution

In this section, the software maintenance and evolution paradigms are studied. That is to define them, differentiate their purpose and work, and explain their need and utilization in this thesis work. Moreover, the examination of the established work to support developers performing evolution tasks was considered important, as this thesis work intends to aid engineers and developers in performing maintenance tasks and decisions.

2.7.1 Fundamental Paradigms and Processes

Software maintenance refers to the software life cycle phase beginning when the first delivery of the software is made, and ending when the software is taken to close-down. On the other hand, **software evolution** refers to the step-wise incremental development of the software during its lifetime. Evolution of the software system takes place both in the development and maintenance phases through successive and concurrent changes. The activity of maintaining these changes is called **change management**. Evolutionary software development is a process in which the software is delivered incrementally (Bennett and Rajlich 2000) (Rajlich 2014). During the maintenance phase, the continuous customers' feedback through bug reports and change requests generates the requirements for subsequent deliveries.

A very widely cited survey study in (Lientz, Swanson et al. 1978) (Lientz and Swanson 1981) , and repeated by others in different domains, exposed the very high segment of life-cycle costs that were being expended on maintenance. The authors categorised maintenance activities into four classes:

1) Adaptive - changes in the software environment
2) Perfective - new user requirements
3) Corrective - fixing errors
4) Preventive - prevent problems in the future.

These studies show that the incorporation of new user requirements is the core problem for software evolution and maintenance. For this motive, this research emphasises the importance of employing requirements' models and linking it to customers' feedback to ensure traceability and that requirements information and dependencies are kept-up-to-date, which are key factors for an enhanced and more accurate maintenance decisions (i.e. Perfective Maintenance). Furthermore, this research stresses on the importance of capturing contextual information as a main component in customers' feedback, which helps in adapting software to changes in the environment (i.e. Adaptive Maintenance). Moreover, acquiring users' feedback about the system helps in resolving the issues they encounter (i.e. Corrective Maintenance). Also, linking feedback to features and business goals can help engineers benefit from past experiences (by analysing feedbacks and their links to models) and plan ahead (i.e. Preventive Maintenance). Thus, addressing these problems through a systematic process will ensure the capturing, analysis and reusability of information that can favour the four maintenance categories mentioned above.

System change requests are the key driver for system evolution in all organizations. These change requests may involve bug fixes to existing requirements, enhancements, new features, or feature proposals by engineers to their customers as a suggestion to improve their work. As illustrated in Figure 3 below that the change identification and software

evolution process are cyclic and continuous throughout the lifespan of the system.

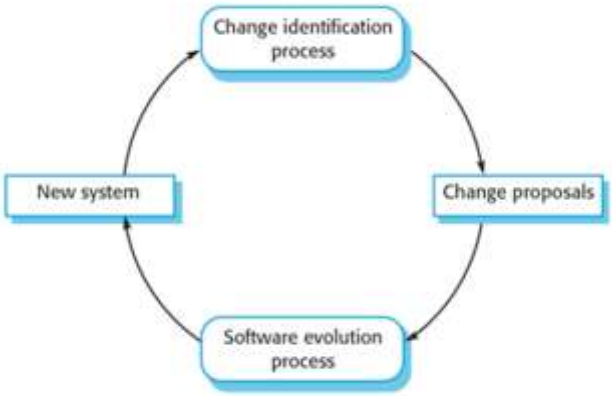

Figure 3. Change Identification and Evolution Processes (Sommerville 2006)

Furthermore, the evolution process in Figure 4 includes the fundamental change management activities which are analysing the impact of a change, planning for the change, change implementation, then system release and closure.

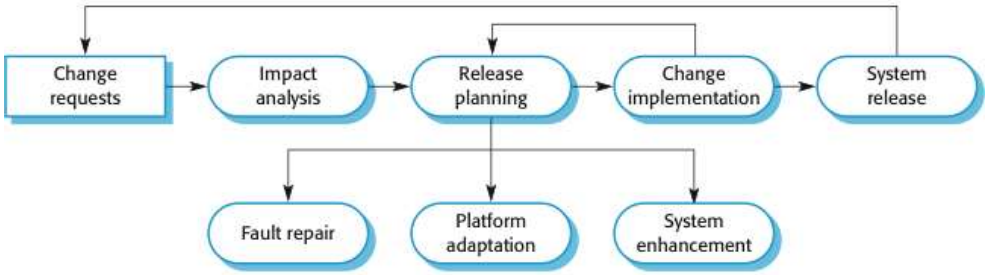

Figure 4. The System Evolution Process (Sommerville 2006)

This research focuses on the capturing and utilization of end-users feedback to inform the maintenance and evolution tasks. Thus, studying these processes of change management and evolution is crucial, as it would be beneficial to align the new process of feedback acquisition, communication and requirements extraction and documentation resulting from this research to well-established processes that practitioners and researchers are accustomed to utilizing them in the projects they work on. This could help developing their mind-sets and gaining their acceptance to easily adopt the new suggested methods for handling and utilizing feedback.

Also in (Sommerville 2006), pointed that change management support tools should provide some or all of the following facilities:

- A form editor that allows change proposal forms to be created and completed by people making the change requests.

- A workflow system that allows the change management team to define who must process the change request form and determine the order of processing. Also, this system will automatically pass forms to the right people at the right time and inform the relevant team members of the progress of the change.
- A change database that is used to manage all change proposals. Database Query facilities allow the change management team to find specific change proposals.
- A change reporting system that generates management reports on the status of the change requests that have been submitted.

This was studied in order to identify the expected key components of the proposed engineering method and framework. Also, in order to help in mapping and evaluating the benefits provided by the framework, by describing how the framework contributes to each of these facilities.

2.7.2 Supporting Developers in Software Evolution Tasks

In this section a review of the work done in the literature in attempt to support developers in their software evolution tasks has been made. In order to devise new mechanisms that could help engineers on their evolution tasks, the tasks they do should be identified which was discussed in section 2.7.1, and their needs and the questions they ask should be investigated in order to recognize the gaps and attempt to produce desirable solutions.

Although fundamental processes and tools are available to guide and help programmers on working on change and evolution tasks, and several studies have been conducted to understand how programmers comprehend systems and requirements (Phalp, Adlem et al. 2011), little is known about the specific kinds of questions programmers ask when evolving a code base.

To fill this gap (Sillito, Murphy et al. 2006) have conducted two qualitative studies of programmers performing change tasks to medium to large sized programs. The developers' experience ranged from newcomers to industrial experienced developers. The results they came up with helps in gaining a deep understanding of the developers' needs and scope during the evolution tasks. However, their work investigates the developers need on the code level. However, this research focus is on the requirements' evolution and update, which also affects the analysis and implementation of new changes.

Other attempts to guide developers in performing evolution tasks is (Zimmermann, Zeller et al. 2005) work. The authors also work on the code level and employ the data mining technique to version histories in order to guide programmers along future related changes. This research shares same interests but using different set of information (i.e. feedback threads), as by having stored historical feedback threads, it could be analysed to help the engineers predict customers' changing needs and plan ahead.

Still more attempts to support the developers on the code level were made by (Würsch, Ghezzi et al. 2010). The authors suggest modelling the data using OWL ontology and use knowledge processing technologies from the Semantic Web to query it. Thus they do not only rely on predefined query. Instead, the querying capabilities of their framework are much more flexible and extendible due to the use of ontology. This is has a great deal of similarity with how this research intends to formalize the research finding and put them into practice. This could be achieved by defining ontology for the structured feedback types and their constituents and thus all feedback threads could be stored on the ontology knowledge base for further querying to inform the engineers' evolution tasks and decisions as suggested in (Sherief, Abdelmoez et al. 2015).

2.8 The Use of Ontologies in Requirements Engineering

In recent years the development of ontologies, which is the explicit formal specifications of the terms in the domain and relations among them have become common in many fields. For example, the ontologies on the Web range from large taxonomies categorizing Web sites, such as on Yahoo!, to the categorizations of products and their features, such as on Amazon.com. Ontologies define a common vocabulary for researchers who need to share information in a domain (Noy and McGuinness 2001). It includes machine-interpretable definitions of basic concepts in the domain and relationships among them. In contrast to other data formats and associated tools, such as XML and XQuery2 that operate on the structure of the data, OWL enables treating of data based on its semantics. This allows the simple extension of the data model while maintaining the functionality of existing tools.

The introduction of ontologies as a means to define the information and knowledge semantics become more and more accepted in different domains. The nature of requirements engineering includes gathering knowledge from various sources, which includes many stakeholders with their own interests and points of view. Therefore, there are many possible usages of ontologies in Requirements Engineering (RE).

(Happel and Seedorf 2006) present methods for using ontologies in the area of Software Engineering with specific focus on analysis, design and implementation phases of the software development life cycle. Each method was defined regarding the problem it attempts to resolve. It is followed by a short description of the method and the assumed advantages of ontologies. They state that ontologies seem to be well suited for requirements management and traceability. Also, formal specification may be a prerequisite to comprehend model-driven approaches in the design and implementation phase.

In (Castañeda, Ballejos et al. 2010) discuss the different challenges faced during the RE process and the benefits of ontologies in addressing the identified challenges. They propose an Ontology-based framework for supporting semantics based requirements engineering.

These proposals can be clearly divided into three application areas, which are: the description of requirements specification documents, the formal representation of the application domain knowledge, and the formal representation of requirements. Although the paper moves the field forward by demonstrating the importance of implementing ontologies in certain circumstances and RE activities, more work is still needed in order to produce a more integrated framework, capable of tackling the classified challenges in an integrated way, and of being generically applied all over the RE process and its activities.

(Siegemund, Thomas et al. 2011) identified weaknesses of RE methods and tools that can be summarized as follows: 1) Requirement knowledge is not adequately covered. Purposes, risks, problems and choices are not documented during RE and consequently, are not available at later stages during software development; 2) Relationships between requirements are ineffectively gathered and are frequently restricted to binary relations between requirements; 3) Requirement problems (e.g. conflicts, unspecified information) are detected too late or not all; 4) Completeness and consistency are not validated. They also introduced the idea to use ontology for structuring the concepts, requirements and relationships captured during requirements elicitation.

2.9 Controlled Natural Languages

In this research the need for structuring feedback and its benefits was argued. The structure of feedback includes two perspectives. The first is identifying the elements that constitute the feedback to enable its formalization and systematic use. To further formalize the feedback, the second perspective that can be considered is the textual writing foundations that could be utilized to provide well-written formal feedback.

This research suggests utilizing the collected feedback during the communication between end-users and engineers in extracting information for keeping requirements specification up-to-date. Thus, comprehension is indeed an important goal to consider for the accurate documenting of requirements (Phalp, Adlem et al. 2011).

Controlled natural languages (Kuhn 2014) are rich subsets of natural languages (i.e. Standard English), obtained by controlling the syntax and semantics in order to reduce or remove ambiguity and complexity. Controlled languages fall into two major types: those that improve clarity for human readers, and those that enable consistent automatic semantic analysis of the language. In this research the utilization of Controlled Natural Language is suggested to merge between these two types, because the language will restrict the user by general rules such as keeping sentences short, only use the reserved keywords to define textual blocks. But also, it will have a formal basis.

In order to construct an acquisition method in this research an already existing controlled natural language may be adopted, namely Attempto Controlled English (ACE) (Fuchs,

Kaljurand et al. 2006) that will act as our text writing foundation that users will use to write their feedbacks more precisely. It is proposed for professionals who want to use formal notations and formal methods, but may not be familiar with them. Though ACE appears absolutely natural – it can be read and understood by any speaker of English – it is in fact a formal language. ACE and its related tools have been used in the fields of software specifications, theorem proving, text summaries, ontologies, rules, querying, medical documentation and planning.

This language includes: the definition of the language Syntax Construction Rules which is the set of rules and principles that control the structure of sentences in the language, and the definition of the Interpretation Rules that deterministically interpret syntactically correct sentences in users' feedback.

2.10 Recommender Systems

This area (Adomavicius and Tuzhilin 2005) in the literature was reviewed, because in this thesis it is argued that having structured and valid feedback is a very useful premise to extract meaningful information about the feedback subject and its relationship to other feedbacks. This work relies on users to give feedback which is going to be used for maintenance, adaptation or evolution decisions. Recommender systems techniques can help find similarities: 1) between features, 2) between feedbacks, and 3) between users. For example, these techniques could be benefited from in reusing feedbacks that have verified solutions and new problems entered by users. This can provide value for users by reusing existing feedbacks to suggest verified mitigations that they can try in order to resolve their issues.

2.10.1 Content-based filtering Recommender Systems

One approach when designing recommender systems is content-based filtering. In a content-based recommender system, keywords are used to define the items; beside, a user profile is built to specify the type of item this user likes. In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended.

2.10.2 Knowledge-Based Recommender Systems

Knowledge-based recommender systems rely on explicit knowledge about the item features, user preferences, and recommendation criteria (such as contexts). The advantage of this approach over previous ones is the removal of cold-start problems. However, eliciting and shaping the knowledgebase needs careful effort and study, which is subsided by the acquisition method that will designed, which will enable structured and validated feedbacks

to be modelled and stored in the knowledge base.

2.10.3 Hybrid Recommender Systems

Hybrid approach makes use of both collaborative filtering and content-based filtering in order to achieve a better result. This implementation can be achieved by combining the results from each filtering methods, or by mixing them into a single model. Several studies have proven that a hybrid approach towards designing a recommender system can lead to results that are more accurate. For example, Netflix, they make recommendations by matching the watching and searching behaviours of similar users (i.e. collaborative filtering) as well as by proposing movies that share characteristics with films that a user has well-ranked (content-based filtering).

2.11 Summary

This chapter presented a review of the state of the art in relation to the user involvement in different application areas specifically requirements and evolution tasks, the acquisition of users' feedback in software applications and potential approaches to impact the solution space of this thesis. The next chapter explains the three approaches to research, and explains in detail the reasons for choosing adopted research approach, design and methods, and analysis techniques.

3. Research Methodology

The overall research procedure starts from choosing the research approach, research designs, and specific research methods of data collection, analysis, and interpretation. Therefore, in this chapter a discussion of the research design and methodology will be made. Firstly, it is important that different views are analysed. Afterwards, the selected methodology will be discussed. This is to provide the plan for the research, and to verify the validity of the final results.

3.1 The Three approaches to Research

Research approaches are plans and the practices for research that cover the steps from broad assumptions to detailed methods of data collection, analysis, and interpretation. This plan involves several choices that need not be taken in the order to achieve the research objectives, and answer the research questions. The selection of a research approach is also influenced by the type of the research problem or the concerns focused on, the researchers' personal skills, and the audience for the studies. In this section, the different approaches to research are explained by defining each type, stating its main characteristics, and possible disadvantages. Finally, the selection of a specific approach for this research is discussed.

3.1.1 Qualitative Research Approach

Qualitative Research is mainly exploratory research (Berg 2004). It is used to gain an understanding of core reasons, opinions, and motivations about a research area or problem. It provides insights into the problem and helps to develop ideas, conclusions or hypotheses for potential quantitative research. Qualitative research is designed to uncover a target audience's range of behaviour and the perceptions that drive it with reference to specific topics or issues. It uses in-depth studies of small groups of participants to guide and support the construction of hypotheses. The results of qualitative research are descriptive rather than predictive.

The main characteristics of qualitative research (Creswell 2013) are: 1) it aims to studying real-world situations as they evolve unaffectedly; 2) the researcher is open to whatever insights that arise (i.e., there is a lack of pre-set constraints on findings); 3) the researcher avoids rigid designs that eliminate responding to opportunities to look for new paths of findings as they arise; 4) cases for study (e.g., people, organizations, communities, cultures) are selected because they offer useful insights of the topic of interest; and thus sampling is aimed at insight about the topic, and not experimental generalization derived from a sample and applied to a population;

Some possible weaknesses related to using qualitative methods to study research problems

in the social sciences include (Berg, 2004): 1) Moving away from the original objectives of the study in response to the changing context under which the research is conducted; 2) Arriving at different conclusions based on the same information depending on the personal knowledge and characteristics of the researcher (researcher bias); 3) Data collection and analysis is often time consuming and/or expensive; 4) Researcher Bias can enter in the data collection and analysis; 5) Requires a high level of experience from the researcher to obtain the aimed information from the participant.

3.1.2 Quantitative Research Approach

On the other hand, **Quantitative Research** (Creswell 2013) is used to quantify a problem by generating numerical data or data that can be transformed into utilizable statistics. It is used to quantify attitudes, opinions, behaviours, and other defined variables – and generalizes results across a larger sample population or to explain a particular phenomenon. Quantitative Research uses measurable data to formulate facts and uncover patterns in research.

The main characteristics of quantitative research are (Kitchenham, Pfleeger et al. 2002): 1) The data is usually gathered using structured research tools; 2) The results are based on larger sample sizes that are representative of the population; 3) The researcher has a clearly defined research question to which objective answers are sought after; 4) All aspects of the study are carefully designed before data is collected; 5) Data are in the form of numbers and statistics, often arranged in tables, charts, figures, or other non-textual forms; 6) Can be used to generalize concepts more widely, predict future results, or investigate underlying relationships.

The results of quantitative research may be statistically significant but are often humanly inapt. Some specific drawbacks associated with using quantitative methods to study research problems in the social sciences include (Berg 2004): 1) Quantitative data may lack contextual detail; 2) Results provide less detail on behaviour, attitudes, and motivation. Thus, the answers may not effectively convey how people feel about a topic or issue and in some cases, might just be the closest match to the predetermined hypothesis; 3) Results provide numerical descriptions and thus generally provide less deep explanations of human opinions; 4) The research is often carried out in an unrealistic environment so that a level of control can be applied to the exercise, and thus, leading to lab results instead of realistic results that could be applied in the real world.

3.1.3 Mixed Methods Research Approach

Finally, the **Mixed Methods Research** (Creswell 2013) is an approach encompassing the collection of both quantitative and qualitative data, integrating the two forms of data, and using distinct designs that may involve hypothetical assumptions and theoretical structures.

By mixing both quantitative and qualitative approaches, design and methods, the researcher gains in breadth and depth of understanding and validation, while offsetting the weaknesses incorporated to using each approach by itself. Yet, the researcher must make a decision regarding the implementation sequence of data collection methods.

Mixed methods research is specifically suited when the researcher wants to (Creswell 2013): 1) validate the results obtained from other methods; 2) needs to expand, clarify, or build on findings from other methods; 3) look at a research question from different angles, and clarify unexpected findings and/or potential contradictions; or 4) to generalize findings from qualitative research.

However, there are several disadvantages associated with undertaking a mixed method approach to research, which includes (Creswell 2013): 1) the research design can be very complex and expensive; 2) It may be time consuming and require many resources to plan and conduct this type of research; 3) may be difficult to link or plan a one method based on the results obtained from another; 4) it may be unclear how to resolve the inconsistencies that occur during the analysis and interpretation of the findings.

3.1.4 The Adopted Research Approach

A qualitative approach was chosen to this research. Firstly, the research objective was to study the readiness and willingness of end-users to take a more active role in feedback acquisition, modelling, and also interfacing in the sense of how they expect a tool support for such purpose to be like. Conventional software engineering processes lack a common theory for the collective involvement of users and their communities. There was no solid background in the literature about this area to base the research upon. Thus, an exploration about the end-user involvement perspectives related to their roles, behaviours, knowledge, and personal experiences, and also their issues and concerns was needed. This initial study led to the evolvement of a set of themes (i.e. topics to consider) regarding the investigated research area, where each could be further examined and studied from different angles.

Secondly, it was considered for the best interest of this research to conduct more in-depth study of the feedback structure and its constituents, which was one or the main concluded themes in the initial study, in order to develop a solid foundation that will enable this research to move forward towards inventing a novel software engineering process for feedback acquisition and modelling. Thus, more exploration was needed in order to gather insights on how the end-users in the communities of business software provide and respond to feedback, and utilize it in resolving similar issues or situations. A novel classification of feedback, their constituents and relations was reached.

Since this research aim was to acquire and make use of the end-users' feedback at during

the software maintenance and support phase while the software is in use, it was intended to involve the role of software engineers, as they will be reviewing the end-users' feedback in order to plan for updates or next versions of the software. Thus, a further investigation was needed to examine from their perspective the current feedback communication handling methods with end-users in the evolution process and its associated problems. Also, to explore from the engineers' perspective the usefulness of the novel classification of feedback, their constituents and relations that was reached, and how it could be utilized to inform the evolution process tasks, and extract requirements information updates.

At the final stage of this research there was a need to integrate the concluded phenomena from the previous studies to design the intended outcome of this research. That is to design and develop a software engineering process for feedback acquisition, communication, and documentation. The Participatory design research, which is considered a qualitative research methodology option, was chosen to achieve this part of the research. It involved both actual end-users and software engineers in the design process, and helped the researcher observe how the prototypes were practically used by them. Also, participants were active in making informed decisions throughout all aspects of the research process. This helped validate the designed outcome in practice as it evolved from participants' needs and opinions.

3.2 The Research Design

The researcher not only selects a qualitative, quantitative, or mixed methods approach to undertake, but also decides on a type of study within these three choices. Research designs are types of inquiry within an approach that provide specific direction for procedures in a research design.

3.2.1 Qualitative Research Designs

Since it was explained in the previous section that a qualitative approach was chosen for this research, this section will focus on the qualitative research designs. And justify the selected design for this research. There are five main **qualitative research designs**, which are categorized as follows (Creswell 2007) (Lazar, Feng et al. 2010):

1) Narrative: In depth investigation of someone's story in order to gather data about what the story means and the lessons learned from it.
2) Phenomenology: Studying participants' experience about phenomena in a certain context that they lived to generate explanations.
3) Grounded Theory: investigates individuals' interactions and views of the problem rather on depending on prior hypotheses with the goal of developing a theory.
4) Ethnography: is an in-depth description of a people group done through participant's

observation to discover a “cultural” phenomenon or pattern. It is then recorded in the language of the host society under investigation.

5) Case study: The most common type of qualitative research, a case study looks at irregular events in a certain context. The overall purpose is generally to explain “how” by explanation.

3.2.2 The Adopted Research Design

The Grounded Theory was one of the most appropriate approaches to take. Grounded Theory is an inductive methodology, meaning that it allows researchers to discover as much as possible variations in people’s behaviours, issues and/or concerns about the problem to generate new theories from data rather than depending on prior hypotheses (Lazar, Feng et al. 2010).

Therefore, it can be defined as the systematic generation of theory from systematic research. It is a set of rigorous research procedures leading to the emergence of conceptual categories. These concepts/categories are related to each other as a theoretical explanation of the action(s) that continually resolves the main concern of the participants in a substantive area.

Since the Grounded Theory (Creswell 2013) approach is being adopted in this research then the 4 –stage Grounded Theory analysis method will be undertaken in this research. Once the data is collected, a series of Codes will be created from this data (i.e. user quotes or observations by the researcher) to allow the key points of the data to be gathered. Codes with similar content will be then grouped into Concepts to make the data more meaningful and workable. Finally, broad groups of similar concepts will be grouped into Categories which will be used to generate theory which is a collection of explanations that explains our subject of research.

This research design specifically suited this research area and topic, because it was a green area where research ideas were still evolving and considered from different angels. And thus there was no solid background to build upon or derive hypotheses. Accordingly, more exploration was needed to generate ideas that could help direct and narrow down this research, and to obtain more data about behaviours, concerns and issues of both end-users and engineers to help in devising new methods for feedback acquisition, communication, and requirements extraction that could help each in his side/role.

3.3 The Designated Research Methods

In this section the research design process will be discussed along with the research methods that were used to conduct the studies.

3.3.1 Thinking about Users and Design

"There is no direct path between the designer's intention and the outcome. As you work a problem, you are continually in the process of developing a path into it, forming new appreciations and understandings as you make new moves." (Charlotte Magnusson 2009)

It was thought that since the end-users were centres of this research, meeting real users and exploring real situations, will have a major impact on the way the work will be carried out and evolve. Moreover, another important role in this research was the software engineers, who will judge whether the results reached from the end-users perspectives are meaningful and useful to inform their maintenance tasks and decisions. So meeting with them was also essential for this research design too, as the aim was to capture both perspectives and include them in the design.

There are three basic components in any design process (Charlotte Magnusson 2009):

1) Idea generation: Ideas should be generated, selected and visualized (articulated). This was done through a two-stage study from users' perspective, where the first study was carried out to generate ideas about end-users needs, issues and concerns regarding feedback acquisition and communication, which helped narrow down the research path. The ideas (themes) were visualized using thematic maps which represented the core themes surrounding the main idea and the inner categories and concepts regarding each theme. The second study was carried out where one of the initial themes was selected and more examination was needed to capture the core feedback types, their constituents, and relations. Detailed explanation of the research methods and study results is made in chapter 4.
2) Know the user, usage and context. You should try to discover user needs, how the user performs the same set of tasks today, how the user will use the proposed artefact, and how this fits into the context.

   Partially this was achieved in the first two-staged study process explained in the previous step which focused on the end-user perspective. Furthermore, this was achieved through a third study that was conducted with software engineers in a business software house. Software engineers were also targeted users in this research. Thus, an investigation of their current issues while communicating with end-user using feedback and how this affects their maintenance tasks and decisions was needed. Also capturing their needs, and expectations regarding the utilization of the developed feedback structures in the communication, maintenance tasks, and requirements updating was essential. Detailed explanation of the research method and study results is made in chapter 5.
3) Evaluate. Ideas, concepts, models, prototypes need to be evaluated.

   A fourth and final study was conducted, where ideas, concepts, models, and

prototypes were materialized and put into practice with both the end-users and engineer. An in-depth study was conducted were a feedback acquisition prototype was designed, and an initial design for feedback acquisition and communication engineering process was designed too. Both designs were put into practice by immersing the participants in fictional scenarios that imitate real situations to help them evaluate and evolve the designs. Furthermore, another method was designed with engineer participants to handle the updating of the utilized feature models, and create/ update feature specification documents. Detailed explanation of the research method and study results is made in chapter 6.

These activities are not strictly separable. To be able to visualize or articulate the researcher needs to know the users (types and roles) and their usage. And information about the needed user and the usage involved in the research may result from the evaluation of visualized ideas or concepts.

3.3.2 Choosing and Combining Techniques

In the previous section the research studies were discussed from the perspective of why they were needed and how they were used to achieve the research aims of this research. However, no details about the research methods used for data collection and interaction with the decided users and/roles of this research were mentioned or discussed. In this subsection, a brief walkthrough the methods will be made without going into much details which are explained in a separate chapter of each study (chapters 4,5 and 6).

No single technique will be appropriate for all needs and situations. Because of this a researcher needs to discover and plan a suitable set of techniques that will go well together. The basic factor to consider is what kind of input is needed, that is to decide what needs to be achieved. For example, whether ideas are being explored, or design being evaluated. After this start looking at which techniques that can be expected to help in achieving the intended goals. There are some techniques more suitable for idea generation, while others are more targeted at evaluation (Charlotte Magnusson 2009).

In this research the first conducted study employed the focus groups technique (Berg 2004) (Lazar, Feng et al. 2010) where two focus groups were conducted with 15 end-user representatives. Focus groups are a well suited research technique for ideas generation to provide valuable input early in the design process, due to the high degree of user interaction and brainstorming. However, it needs to be considered that usually the study takes place out of realistic context.

Thus, a forums' analysis study (Marra, Moore et al. 2004) was conducted in order to analyse how users provide feedback in actual contexts. Three online forums for business software were chosen, representative active feedback threads were selected and analysed. This

study was planned with a more in-depth perspective, which is trying to examine real end-users' feedback to come up with a more concrete description of feedback, its components, and interrelations.

Moreover, for exploring the software engineers' perspective a study was conducted in a business software house, where 10 participants from 4 different roles were interviewed (Lazar, Feng et al. 2010). The aim of the study was to explore with them how they carry out the change management an evolution tasks, what are the current challenges they encounter that hinder their ability to efficiently perform their tasks and take decisions, and how they could utilize the novel feedback classifications. Interviews are one of the most traditional techniques for exploring requirements. Even if the general procedure for doing interviews (Berg 2004) is rather straightforward, there are many techniques that are useful to learn to carry out the interviews in a professional, efficient and in a context scientific way. This was ensured through conducting an initial introductory session where the topic was introduced, the results reached so far, and the purpose of the study. For the research results, prepared materials with the feedback types list and examples from the forums threads to illustrate how they were derived were presented to the participants. This was to help them understand the results better from real cases similar to what they encounter in their work, and conceive an idea about how the results could be utilized. Furthermore, for pragmatic reasons a confirmatory interviews study was conducted with five other participants who come from different back grounds. This is to confirm the results reached with them and share their perceptions about any particular situations or gaps they determine.

In this research a qualitative "bottom-up" research approach was taken, which means that concepts, needs, and challenges all evolved from actual stakeholders in the research studies. That is why a need for a separate validation study will be waived. Instead, at the end, the proposed approach will be verified and validated by applying it in practice through a participatory design method conducted with both end-users and engineers to 1) investigate the use of the novel feedback types in practice and whether they are easy-to-use by end-users. 2) Also, to investigate whether they are successful means in providing engineers with useful/ meaningful information that could help them in accomplishing the evolution tasks and taking decisions through the new proposed approach for feedback acquisition and communication. 3) And to design with the engineers how the stored communication threads could inform the requirements extraction and updating method.

A participatory design method (Kensing and Blomberg 1998, Spinuzzi 2005, Foth and Axup 2006) was chosen in order to assist in designing the intended approaches which are 1) the design of feedback acquisition, communication and requirements updating method and evolve the initial process designs and tool prototypes with their intended audience (i.e. the end-users and the software engineers) (Kanstrup 2012). Thus, a verified design outcome is ensured; 2) and to design with the engineers (i.e. the intended users of the approach) a

method for extracting requirements information and updating the feature model and specification.

Furthermore, triangulation (Jick 1979) is a method used in qualitative research that involves cross-checking multiple data sources and collection procedures, which is the case in this research as several studies were conducted using different methods, so evaluating the extent to which all evidence converges was necessary. Qualitative analysis of text is often supplemented with other sources of information to satisfy the principle of triangulation and increase trust in the validity of the studies' conclusions. This was also one of the motives and advantages for choosing the participatory design method in the final study of this research. Figure 5 shows the mapping between the research objectives and the research methodologies used to achieve them.

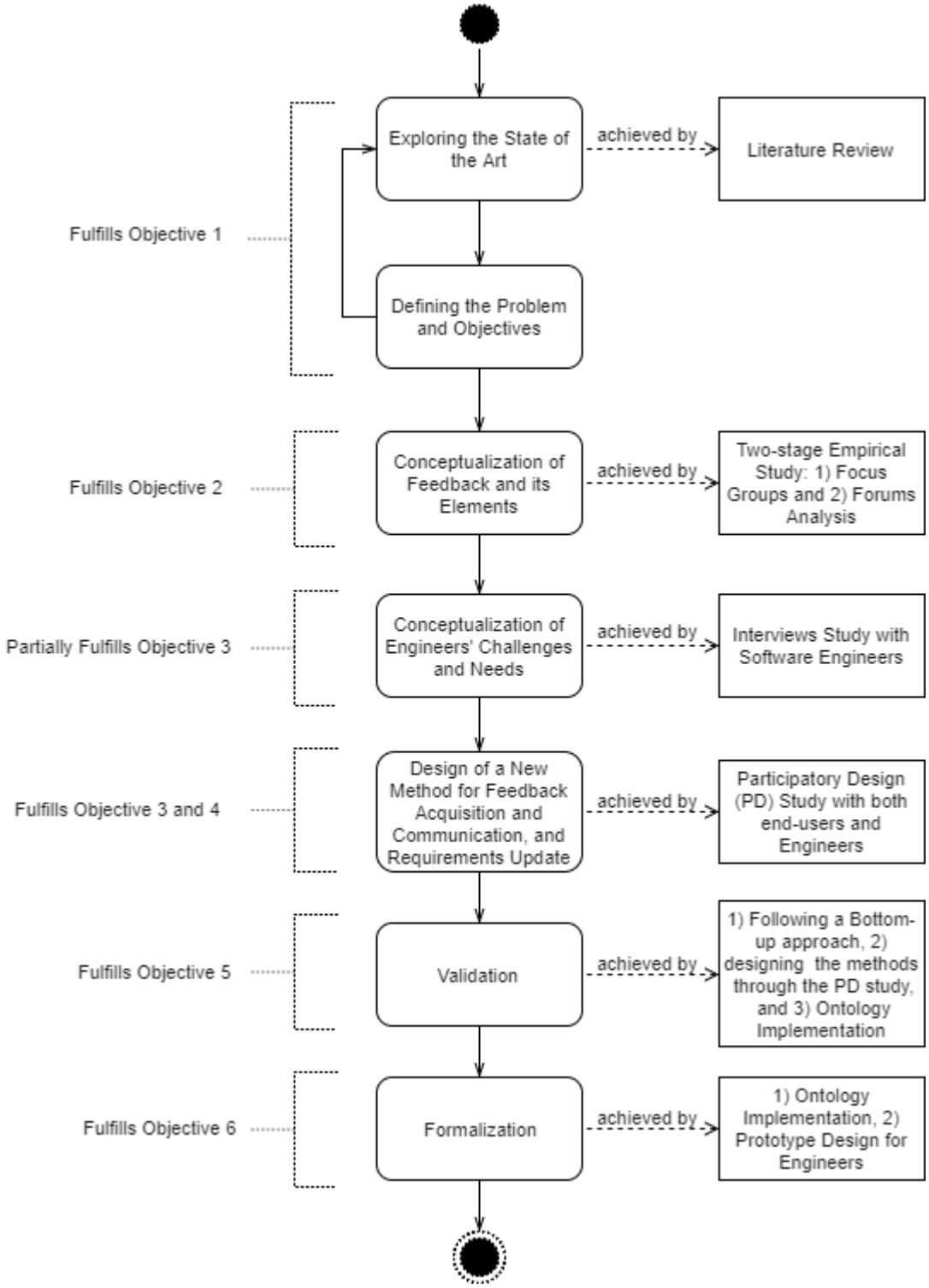

Figure 5. The Mapping of the Research Objectives, the Research Process and the Adopted Research Methodologies

3.4 Thematic Analysis and Content Analysis

In this section the two main qualitative data analysis techniques will be discussed, which are content analysis and thematic analysis (Vaismoradi, Turunen et al. 2013) (Joffe and Yardley 2004). The definitions and usages for both techniques will also be explained. Then the

adopted thematic analysis approach will be discussed.

3.4.1 Thematic Analysis

Thematic analysis is often seen as a poorly branded method (Vaismoradi, Turunen et al. 2013), in that it does not appear to exist as a named method of analysis in the same way that content analysis does. Thematic analysis as an independent qualitative descriptive approach is mainly described as “a method for identifying, analysing and reporting patterns (themes) within data” (Braun and Clarke 2006).

A main question to deal with in terms of coding is what is considered a pattern/theme? The answer is “A theme captures something important about the data in relation to the research question, and represents some level of patterned response or meaning within the data set.” Themes or patterns within data can be recognized in one of two fundamental ways in thematic analysis: in an inductive or “bottom up" way, or in a theoretical or deductive or “top down" way (Boyatzis 1998) (Fereday and Muir-Cochrane 2006).

On one hand, an inductive approach means the themes identified are strongly linked to the data themselves (as such, this form of thematic analysis bears some similarity to grounded theory). Inductive analysis is a process of coding the data without trying to fit it into a pre-existing coding frame (Braun and Clarke 2006), or the researcher’s analytic preconceptions. In this sense, this form of thematic analysis is data driven. As mentioned in this chapter the researcher had no preconceived idea about the

On the other hand, a “theoretical" thematic analysis would tend to be driven by the researcher’s theoretical or analytic interest in the area, and is thus more explicitly analyst-driven. This form of thematic analysis tends to provide less a rich description of the data overall, and more a detailed analysis of some aspect of the data.

From the advantages of using thematic analysis approach (Braun and Clarke 2006): 1) It has a large margin of flexibility for researchers; 2) Useful method for working within participatory research paradigm, with participants as collaborators; 3) Can usefully summarise key features of a large body of data; 4) Manageable to researchers with little or no experience of qualitative research; 5) Can draw attention to similarities and differences across the data set.

3.4.2 Content Analysis

There are various definitions of content analysis. (Stemler 2001) summarized previous work and stated that content analysis is “a systematic, replicable technique for compressing many words of text into fewer content categories based on explicit rules of coding”. This is a narrow definition as it limits content analysis to the textual information domain. A broader

definition was proposed by (Holsti 1969) which states that content analysis is “any technique for making inferences by objectively and systematically identifying specified characteristics of messages”. According to this definition content analysis data can come in different forms such as books, pictures, ideas, music or videos.

“Content analysis is normally an in depth analysis that searches for theoretical interpretations that may generate new knowledge” (Lazar, Feng et al. 2010). It refers to a data analysis technique used in both quantitative and qualitative research. This technique assists the researcher to identify important data from a data corpus. The aim of the researcher is to analyse the content of each data item. In most content analyses, researchers use coding systems to identify and categorize various data items.

When studying data analysis in conducting research, content analysis and thematic analysis are the most two types of analyses used in research. For most researchers, sometimes the difference between content and thematic analysis can be rather confusing as both include going across the data to identify patterns and themes. However, it is important to emphasize that the key distinction between them is that in the content analysis, the researcher can focus more on the frequency of occurrence of various categories, while in the thematic analysis, it is more about identifying themes and structuring the analysis in the most consistent and organized manner. This is why content analysis is now vastly used in communication and media.

Some practical applications of content analysis (Stemler 2001) are: 1) determining authorship by compiling a list of suspected authors, examine their prior writings, and correlate the frequency of nouns or function words to help build a case for the probability of each person's authorship of the data of interest; 2) Content analysis is also useful for examining trends and patterns in documents; 3) content analysis provides an empirical basis for monitoring shifts in public opinion.

3.4.3 The Adopted Analysis Technique

This work has been based on “good quality“ data corpuses and data sets. This could be judged by a specific set of criteria regarding what, why, and how they were collected, and offer rich, detailed and complex interpretations of the topic. Good data do not just provide a shallow overview of the topic of interest, or simply repeat a common-sense explanation.

The studies conducted in this research were all with potential users (whether customers or software engineers), who deal with the research problems in their daily work. This research used “bottom-up” approach in thematic data analysis, as there was no preconceived idea about the depth and levels of details that the participants were going to provide and thus the analysis was entirely data-driven.

Thematic analysis was seen as the most suited technique to analyse the collected data in this research as it has a margin for flexibility (Braun and Clarke 2006). Moreover, this research has resulted in introducing new concepts and engineering methods to the area. This evolved through the staged studies that were conducted though this research, and therefore it was necessary to use an analysis technique that will assist in providing rich descriptions emphasizing context (rather than quantifiable results) to help term the new concepts and introduce novel methods.

3.4.4 Qualitative Research Analysis Tool

"NVivo is software that supports qualitative and mixed methods research. It's designed to help researchers organize, analyse and find insights in unstructured or qualitative data like: interviews, open-ended survey responses, articles, social media and web content." (International Copyright © 1999-2014)

Without tool support working with qualitative data will be more time consuming, difficult to manage, and hard to traverse. Essentially, completing this kind of research without tool support can make it very hard to determine connections in the data and find new insights.

NVivo can be also used to provide tools that allow researchers to ask questions of their data in a more efficient way. On example, is that it can be used in coding comparison to compare the degree of agreement and disagreement between the analysis content of two different researchers.

3.5 Ethics in the Research

There are a number of key phrases that describe the system of ethical protections that were created to try to protect better the rights of the research participants (Orb, Eisenhauer et al. 2001). The principle of voluntary participation requires that people not be forced into participating in research. Closely related to the notion of voluntary participation is the requirement of informed consent. Essentially, this means that potential research participants must be fully notified about the procedures and risks involved in research and must give their consent to participate. Ethical standards also necessitate that researchers not put participants in a situation where they might be at risk of harm as a result of their participation (Berg 2004). Harm can be defined as both physical and psychological. There are two standards that are applied in order to aid in protecting the privacy of research participants. Almost all research guarantees the participants' confidentiality -- they are guaranteed that identifying information will not be made available to anyone who is not directly involved in the study. The stricter standard is the principle of anonymity which essentially means that the participant will remain anonymous throughout the study. Clearly, the anonymity standard is a stronger guarantee of privacy.

Even when clear ethical standards and principles exist, there will be times when the need to do accurate research runs up against the rights of potential participants. No set of standards can possibly anticipate every ethical circumstance. Furthermore, there needs to be a procedure that assures that researchers will consider all relevant ethical issues in formulating research plans. To address such needs most institutions and organizations have formulated an Institutional Review Board (IRB), a panel of persons who reviews grant proposals with respect to ethical implications and decides whether additional actions need to be taken to assure the safety and rights of participants. By reviewing proposals for research, IRBs also help to protect both the organization and the researcher against potential legal implications of neglecting to address important ethical issues of participants.

In this research, two research ethics submissions were sent to The Bournemouth University Research Ethics Committee (UREC). UREC considers ethical issues related to research and research-related activities brought to its attention by Academic Schools, researchers and the wider university community (University 2017). The first ethics submission covered the focus groups study with end-users, the forums analysis, and the interviews study with software engineers, while the second ethics submissions covered the participatory design study with both the end-users and the software engineers. Both ethical submissions were approved by UREC.

Besides the required ethics checklists two main documents were prepared and submitted for each study. The first is the participant agreement form, which was used in the studies to obtain the participants' signed consent to be involved in the study. The second is the participant information sheet, which briefly introduces the topic to the participants, explains the purpose of the study, clarifies to the participants why they have been chosen, what does taking part of the study involve, and what are the possible advantages, disadvantages, or risks of being involved in the study, also it describes the type of information sought from them. Samples of both documents are available in Appendix 1.

3.6 Summary

In this chapter the three approaches to research were explained, along with details regarding the reasons for choosing adopted research approach, design and methods, and analysis techniques. In the next chapter the first two studies that attempt to achieve the research aims and objectives and introduced. The adopted research methods are explained in more detail, and the results are illustrated and described.

4. Exploring Feedback Structure - User View Point

This chapter explains the two user studies that were conducted in order to explore and observe the structure of users' feedbacks and other aspects related to feedback acquisition. The chapter explains the research methods that were followed in both studies and the results reached from each study. These results act as the foundation for our next steps in the research by providing illustrated results from actual users' opinions and feedbacks.

In this chapter the research methods used to conduct a two phase empirical study are explained. A qualitative approach is adopted (Berg 2004). The study consists of two phases including two focus groups in the first phase and three forums' analysis in the second. Each study will be described, its goals, research questions, and data analysis method, and the themes resulting from each study.

4.1 First Phase Study (Focus Groups)

A qualitative approach was adopted to explore and understand how users provide feedback and their preferences on the acquisition process (Creswell 2013). The study had two phases. In the first phase study a two sessions focus group study was conducted, which is a popular technique of qualitative research in software engineering (Kontio, Lehtola et al. 2004). Focus groups can provide valuable input early in the research process as it is a discussion and demonstration of artefacts (Lazar, Feng et al. 2010). The main advantages of focus groups are: 1) they are useful to obtain detailed information about personal and group perceptions and opinions about the researched topic; 2) identify changes in users' behaviour, how these changes are triggered, and why; 3) they can provide a broader range of information and investigate the use, effectiveness and usefulness of particular services; 4) they offer the opportunity to seek clarification to deepen understanding. The results of the focus groups was analysed using the thematic mapping approach (Braun and Clarke 2006), which is a flexible method for working within participatory research paradigm, with participants as collaborators.

The study goal was to collect some insights and ideas from users who have actually given feedback before, in order to provide us with opinions from their own experience (for more details see Appendix 2). This contributed to our research by providing a set of themes, each that can be further investigated and researched. The main areas to explore were:

1) To explore the ways users would like feedback to look like, and the criteria that judge whether the feedback is meaningful and useful
2) To understand how software users give feedback and how they think a good feedback should be structured.

3) To explore the way users would like to be involved in the process of providing feedback, and what encourages them to act as evaluators
4) To understand how users could benefit from the experience of other users' feedback structures and reuse them to answer similar cases or problems.

The focus group consisted of two separate sessions. A same set of questions were used in each session with different combinations and focuses as summarized in Table 1. The purpose of the study was not to point out the differences between end-users and engineers. But instead to capture a balanced set of concepts that capture the whole picture depending on their roles and experience.

Table 1. Focus group session settings

Sessions	Participants	Purposes
1	Developers who gathered user feedback or got involved in feedback gathering in the past	Channels, forms, expectations
2	Regular software users who provided feedback in the past	Channels, motivations, concerns, experiences

Both junior and senior software developers were invited to join the first session where the emphasis of this session was to understand how software developers normally gather user feedback, how they think a good feedback should be structured and how they collaborate and communicate with users in the development as this could inform the way we design feedback requests. The second session was conducted with regular software users who are used to providing feedback. The emphasis of this session was to explore the ways that users would like feedback requests to look like, what drives them to provide feedback and their concerns for not getting involved enough and also for being involved more than what they expect. This session was also used to investigate their motivations to take part in projects and learn their experience from that participation.

A total of 15 volunteers, 8 males and 7 females aged between 18 and 40, were invited to participate in the two focus group studies. There were 8 participants in the first session (2 females and 6 males) and 7 participants in the second session (5 females and 2 males). These participants mainly came from Egypt and UK with various backgrounds ranging from management, student, research and IT and had different experiences in using software and providing feedback.

It should be noted that most user participants were already familiar with feedback whether they have given their feedback before on a software or product, or used feedback as a way to reach solutions or reuse other experiences. For example, they used feedback for simple

tasks such as collecting the notes for lectures, programming forums to get solutions for certain coding and debugging problems, etc. In addition, it was ensured that all are familiar with the concept by showing demos and discussing main concepts.

Participants of each session were recruited separately following a pre-selection process to ensure they have similar characteristics. For example, for those engineers volunteered for the study, they had to have the experience of gathering user feedback or getting involved in such activities in the past. Similar pre-selection processes were also used in recruiting software users who provided feedback in the past. The same moderator was used for both sessions. The moderator followed a specially designed interview to balance the need for natural conversation and focused discussion when conducting the focus groups.

Each session lasted two hours. All conversations were audio recorded and transcribed with consent from participants. They were aggregated and analysed by using thematic analysis method.

The Focus group was small scale study, so only one researcher acted as the sessions' moderator and primary analyst of the data. Acting as the sessions' moderator, this analyst had full understanding of the data gathered. Furthermore, another researcher looked at the codes to revise them and check if there is any ambiguity. Thus, credibility of the findings was ensured and maximized by the inter-coder agreement and academic advisor's auditing (Miles and Huberman 1994, Creswell 2012).

4.2 Focus Groups Study Results

Following the recommendation of six stages of analysis (Braun and Clarke 2006), four thematic areas were formed and 15 themes were identified from the analysis. The four thematic areas are: subject, structure, engagement and involvement (Sherief, Jiang et al. 2014) . The endpoint is the reporting of the content and meaning of patterns (themes) in the data, where "themes are abstract constructs the investigators identify before, during, and after analysis". The four thematic areas constituting the final thematic map in Figure 7 are explained in details in sections 4.2.2 to 4.2.5, each in a section. Section 4.2.1 shows the initial thematic map.

4.2.1 Initial Thematic Map

Figure 6 shows the initial thematic map that was developed through the first iteration on the focus groups' scripts. Using thematic mapping in analysing the focus groups is a recursive and iterative process, where an analyst can move back and forth as needed throughout the phases. This means that the more the scripts were read the more enhancements, modifications, and codes evolved until the final thematic map in Figure 7 was developed. In essence, coding continues to be developed and defined throughout the entire analysis.

Writing is an integral part of analysis, not something that takes place at the end, as it does with statistical analyses. Basically, codes and categories were analysed, and considering how different codes may combine to form a principal theme. At this phase it is important to use visual representations, such as mind maps to help sort the different codes into themes.

Figure 6. The Focus Groups' Study Initial Thematic Map

4.2.2 Environment Thematic Area

Environment refers to the settings that support users so they feel confident in providing meaningful evaluation feedback. This includes subject specificity, clarity and feedback method.

- In detail, participants would like to use a **method** they prefer to aid them in easily providing feedback.
- Moreover, they would like to give more detailed feedback explanation when they reach a **clear** problem specification. Furthermore, to improve the clarity of feedbacks, participants pointed out that it is preferable to add reasons and explanations in feedbacks to help make their viewpoints more **comprehensive**. Also, providing **structure** to the feedback will decrease misinterpretations and eases the analysis of texts afterwards.
- Subject **specificity** can be goal-oriented, which means by specifying the **quality** attribute in the feedback that concerns the user, such as usability, or reliability. Also, subject specificity can be influenced by the **type** of feedback the user would like to provide, as more users tend to give feedback when they need help or when a problem occurs.

This informs our research (RQ 1 in section 4.1.1), as the feedback acquisition method needs to provide users with software-related terms and/or interfaces in a way they can understand and suits their capabilities, so they feel confident in giving meaningful feedback. Table 2 provides examples of coded phrases for Subject thematic area.

Table 2. The Environment thematic Area

Theme	Codes	Sample Coded Phrases
Specificity	Quality Attribute	*"Specify the quality problem that I am giving feedback about"* *"Feedback should measure the quality of the software"* *"There should be a rate for every quality aspect of the feature. For example privacy, reliability, usability"*
	Feedback Type	*"Feedback when an error occurs, this makes it more accurate (specific)"* *"Users can report a problem and others can suggested a solution to it"* *"It is better if the user can be able to suggest a solution to the problem in the feedback, if he has one"*
Method	Text	*"Add text option to explain details"*

	Audio	*"Voice feedback is fast and an easy way to express opinions"*
	Snapshot	*"It is preferable to be able to take snapshots of the system in order to specify the steps I did in order to reach the problem"*
	Scaling	*"Scaling is easier in natural language and not numbers, such as: good, bad, very bad…"* *"The user should be able to select a feature and specify a scale (rating) while submitting the feedback"*
	Keywords	*"Use a group of keywords provided by the feedback tool to the user to express his feedback"*
Clarity	Structure	*"Using tags or pre-defined keywords can to ease the analysis of text feedback"* *"The feedback has to be hierarchical (in a tree format), where the user starts to navigate according to his answers to questions to reach a specific problem definition"* *"Providing feedback through natural language text can be misinterpreted"*
	Comprehensiveness	*"Giving detailed feedbacks will help in better solution to the problem "* *"I can give more detailed feedback explanation about the problem including when the problem occurs; the alternatives I tried"* *"If I gave the feature rate 7, I would like to add why this rate was given"*

4.2.3 Structure Thematic Area

Structure refers to the attributes of a feedback which are favourable to be seen, mainly, by software engineers. Participants confirmed some common perceptions.

- They thought that feedback would be more useful and accurate if it was **feature oriented**.
- It would be useful to be able to correlate feedbacks according to the **inter-relationships** between the features, because some features may affect the functionality of others.
- It is important to provide the possibility of varying **levels of detail** in the feedback to ensure a minimum level of meaningful and useful information, and also to put into consideration other contextual aspects that might affect the users while giving their feedback.

- Engineers also suggested using simple **measurements** in a way to aid users in giving their feedback through and re-using the experiences of others. For example, users can **rate** how much others' feedbacks were meaningful or useful, and accordingly **statistics** can appear to users to show other useful feedbacks. Also, users can give feedback about their experience with new changes in the software to aid engineers in measuring user **satisfaction**.
- It is also important to consider the **timing** of giving the feedback. Users thought that giving a feedback immediately is important especially in reporting errors or problems, as it helps giving more accurate feedbacks with detailed explanations, and therefore would affect the structure of the feedback.

This informs our research (RQ 2 in section 4.1.1), as feedback structure introduces the challenge of balancing between simplicity and expressiveness of feedback from users who do not necessarily have a technical background but they are still able to give specific and measured feedback when provided with the right tools. So translating users' judgments to terms and language which are perceivable by users and require minimized facilitation of moderators is needed. Having a well-structured feedback will also provide benefits towards a systematic requirements extraction process, and benefiting from other users' experiences which informs our research (RQ 4 in section 4.1.1). Table 3 provides examples of coded phrases for Structure thematic area.

Table 3. The Structure thematic Area

Theme	**Codes**	**Sample Coded Phrases**
Specificity	Feature-Oriented	*"The user choses a component or feature to give feedback about"*
	Inter-Relations	*"Relationships between features should be considered, because a feedback about one single feature may affect other features too"* *"Each block in software may consist of multiple features. Therefore, the user has to specify on which step/feature he is giving his feedback"*
Level of Detail	Depth	*"What is the main feedback body and other additional parts that just explain more about the feedback"* *"Scaling ensures that there is a minimum level of meaningful information that was entered by the user"*

		"if I gave the feature rate 7, I can express why this rate was given"
	Context	*"Feedback is affected by the time I have for giving feedback, or the user's mood while giving the feedback"* *"The device, OS and machine specifications should be entered as a part of the feedback"* *"Feedback should be allowed for users who have been using the system for a while"*
Measurement	Statistics	*"If an old feedback for a software feature states whether it was good or bad, statistics should occur to represent how much it was meaningful and useful or not"*
	Rating	*"Bank of feedbacks, where I can view feedbacks. And rate how much I agree with the feedback or differ from them"* *"The person who posted the problem is the best one to rate which answer is the best (best resolves his issue)"* *"Rating the review if it helped in solving others' problems or not"*
	Satisfaction	*"If the software introduced a change to the user for example a change in the user interface, after a period of usage the user can confirm is it better or worse than before, and a rate bar occurs to express that feeling"*
Timing	Immediate	*"Giving feedback when an error occurs, makes it more accurate (specific)"* *"from the benefits of giving real-time feedback is that you can take live snapshots of the problem that you might not find again later"*
	Delayed	*"In groups sometimes users get admin approval to post after it is too late"* *"Opinion feedback types can be provided offline through an e-mail or form"*

4.2.4 Engagement Thematic Area

Engagement refers to the key merits the acquisition process provides to the involved users that encourage them to take part as evaluators. This includes some key characteristics of

engaged users with the process, and also the qualities that are important to the process. Participants noted some key characteristics of engaged users with the process:

- Participants mentioned that they would like to be **recognized** through their **reputation**. Reputation may be considered as a component of identity as defined by others. Reputation is a vital factor in any community where trust is important. Also, users would take recommendations, and/or solutions into consideration if they are given from **reliable** users. The reliability of users increases the weight of their feedbacks.
- Users like to be **valued** in a way in the participation. Participants mentioned that their feedback is valued by knowing that it taken into **consideration** for further analysis and leads to software enhancements. Also, the possibility to learn from others' experiences provides great value to users as it increases their **awareness** by knowing other possible features variations they were no aware about before.

Participants mentioned that channel and transparency are both important to the acquisition process.

- **Channel** reflects the way users want to interact through feedback. They would like the feedback acquisition process to be simple and **interactive**. Also, after giving their feedback they would appreciate if they can chat with a human expert or with the analyst to **discuss** their feedback.
- It would increase users' trust if they know the cycle in which their feedback will be handled and considered. **Transparency** generally implies openness, which can be achieved in different ways. The user can be **notified** through a message that the feedback will be taken into consideration. Moreover, the user may be notified with the overall **process** of processing the feedback. Also, transparency may be achieved by giving the user an **example** of other users whom their feedback was taken into consideration and their issue was resolved.

This thematic area informs our research (RQ 3 in section 4.1.1), in providing an outline of aspects to consider that increase user willingness to actively participate in such a new role as evaluators and how to support that by software tools. Table 4 provides examples of coded phrases for Engagement thematic area.

Table 4. The Engagement thematic area

Theme	Codes	Sample Coded Phrases
Recognition	Reputation	*"The user's reputation is important, user's feedback with high reputation weighs more as it is more trustworthy"*

	Reliability	*"If the feedback is recommended from a community of trusted and reliable users then yes I would take their recommendations into consideration"*
Value	Consideration	*"Users have to be notified that this feedback will be taken into consideration and that actions will be taken"*
	Awareness	*"The bank of statements or software suggestions may show me features or usages that I was not aware of"* *"If there are other variations of the same feature then I will not care much about ratings and/or opinions"* *"If the user kept using a certain feature without exploring any other ways, the system can increase his awareness by giving him a list of friends' experiences with features"*
Channel	Interactivity	*"The interactions should be very simple and include natural language processing"*
	Chatting	*"Suggests that there could be a community to collect feedback from the users and discuss it with them, and that is to encourage them and show them how valuable their feedback is"* *"Meet analysts to discuss with them problems or enhancements"* *"Solutions in the feedback can be the key to criteria to choose these users for open discussions with the analysts"*
Transparency	Process	*"I should know what will be done with my feedback"* *"it would increase the users' trust and willingness to give feedback if they know the cycle of how their feedback will be used"*
	Notifications	*"When the user submits the feedback, he/she is notified that it will be reviewed"* *"Tell the user that he/she will be notified soon, for example, within 24 hours, so the user knows that within this period an action will be taken"*
	Exemplification	*"Increase users trust by giving them examples of users who gave negative feedback and the issue was resolved"*

4.2.5 Involvement Thematic Area

Involvement refers to a variety of "environmental" aspects that motivate users to participate in the process of feedback acquisition and can directly influence the decisions and activities in using/evaluating the software, which informs our research (RQ 3 in section 4.1.1). Table 5 provides examples of coded phrases for Engagement thematic area.

- **Privacy** issues were raised by participants. Participants differentiated between two aspects in privacy, the privacy of their **identity** "would like to stay anonymous", and the privacy of the **content** they provide (i.e. their feedback) "it is important if the user can control who is able to see his feedback".
- Participants were particularly interested in the **rewards** mechanism for involvement whether through implicit or explicit incentives. **Implicit** incentives are not based on anything tangible. Social incentives are the most common form of implicit incentives. These incentives allow the user to feel good as an active member of the community for example through increasing their reputation. **Explicit** incentives refer to tangible rewards, for examples financial.
- The **level of support** from the feedback system was considered important. Many suggestions were raised about how a feedback acquisition tool can help them. For example, the **interaction styles** "there can be videos to explain to the users what they can do (in order to provide feedback)". The ease of use of the feedback acquisition tool is important. They also suggested that the feedback tool can provide **hints** to the users about its capabilities. Moreover, if there is an **automated** detection in some steps of providing the feedback, this would further ease their job. For example if the tool can automatically detect the feature the user is having trouble with.
- The feedback tool **response** on feedback was also considered important. Two characteristics of system response were discussed, which are the **speed** of response from the system and the **language** of response.

Table 5. The Involvement thematic area

Theme	Codes	Sample Coded Phrases
Privacy	Identity	*"Users would like to stay anonymous (do not specify any info)"*
	Content	*"It is important to ensure privacy of feedback (most important to control who sees my feedback)"*
Rewards	Implicit Incentives	*"Users who gave feedback that positively helped in enhancing a feature can be accredited to the user to increase his reputation"*

	Explicit Incentives	*"If the user has a page for his business and he gave a very good feedback that helped enhance the system. Then we can offer to take care of his business page for free"* *"On freelancing websites, I could award the developers who gave good feedback, by suggesting them to business owners who need softwares"* *"Free trials are to use applications and give feedbacks on the application features"*
Support	Interaction Style	*"The feedback can be interactive that is users can explain how they can add to the software to enhance it"* *"Use of drag and drop components will be easy"*
	Hints	*"Give him hints or a template if the user doesn't know how to give feedback"*
	Automation	*"Try to monitor or record what are the frequent tasks that the users do and their sequence so I can provide shortcuts to them (adaptive and customizable)"* *"For example, the components are used through drag and drop and whenever there is a problem or error in the components connection, the software automatically suggests other ways or components"*
Response	Speed	*"the Software's speed of response to my feedback affects my willingness to give feedback "*
	Language	*"The language of confirming with the user should be friendly (e.g. thank you for your valuable feedback)"* *"The way of asking for feedback should be friendly and not obligatory"*

Figure 7. The Focus Groups' Study Final Thematic Map (Sherief, Abdelmoez et al. 2015)

4.3 Second Phase Study (Forums Analysis)

Our research aims entail building a more concrete description for feedback structures. The focus groups allowed us to get a starting point to that and in order to get the elaborated view. Another study was conducted that involved the analysis of three actual online forums (Marra, Moore et al. 2004) that share the same domain where users give feedback on business software. 200 feedbacks were analysed from different threads found on Microsoft's TechNet, WordPress, and SAP forums.

RQ1) What are the concepts that constitute the feedback structure?

RQ2) What are the relationships/ patterns between feedback concepts?

Forums are a feedback medium where any user can provide their feedback about a product or service (Charlotte Magnusson 2009). Users can also view other users' feedbacks and/or reuse other users' feedbacks and edit feedbacks that they already posted. In this sense, forums gather a variety of user judgements, experiences and practices. For this reason, forums may be a rich source for user problems and user experience with a particular product or service.

Other options were considered such as end-user feedback from the AppStore. A study conducted by (Pagano and Maalej 2013) to study the types of end-user feedback in the AppStore was also examined and studied. Their study shows interesting results on the categorization level of different types of feedback found on the AppStore, without getting into details of what are the constituents of each type. Also, based on their analysis of 1100 user feedbacks on the AppStore, 77.82% of users tend to provide rating feedback type, which is a kind of feedback where users answer static predefined questions and do not elaborate much on their opinions. Thus, extracting the constituents of different feedback types using content from mobile apps would be less effective and lacking the rich content that enables the capturing of the needed concepts.

The type of feedback to be captured, which is the objective of feedback in this thesis work, is the feedback used in the communication between both end-users and engineers while the software is in use (i.e. in the maintenance phase). This input reflects the end-users': opinions about specific features, how the overall software meets their needs and expectations, help requests, bug reports, enhancements requests, new features requests, and also other types that could be used by both parties in their communication. This is to accomplish objective 2 and inform objective 3, which is to provide a systematic manner for feedback acquisition and communication that employs structured feedback that carries useful and meaningful information enough to inform the evolution process tasks and decisions in an accurate and effective manner.

Basically it is intended to undergo a more detailed analysis and observation of users' feedback about software applications used in business environments. Business software was targeted to: 1) avoid the noise typically found in general purpose software, as normally users tend to give a more serious and focused feedback. 2) Also, business users are best fitted from the motivation perspective, because it has a direct value on their work and performance. From a practical point of view, forums will allow us to access a large number of users; take a snap-shot of how users actually give feedback.

Table 6 below shows the software forums used for this analysis study along with a description of the main function they provide for end-users, and the support forums' links. These three forums were chosen in order to target different types of business users with diverse technical capabilities. First, in the TechNet forums, the feedback threads of office users will be analysed. Office is a famous desktop application suite, where users with basic

technical skills use in their simple daily tasks.

Table 6. Targeted Forums data

Forum Name	Product Description	Links
Microsoft's TechNet Forums (Office Suite)	All versions of Office include software for the things users do most often, including working on spreadsheets, or word processing, and organizing business with clear view of e-mails, calendars and contacts.	https://social.technet.microsoft.com/Forums/
WordPress Forums	WordPress is an Open source, and easy to use software that enables users to build websites or blogs.	https://wordpress.org/support/
SAP Forums	SAP (Systems, Applications and Products) is a multinational software corporation that makes enterprise software to manage business operations and customer relations.	http://scn.sap.com/threads

Second, in the word press forums there are different kind of roles (Press 2015) the user can take, such as:

- Super Admin – somebody with access to the site network administration features and all other features.
- Administrator – somebody who has access to all the administration features within a single site.
- Editor – somebody who can publish and manage posts including the posts of other users.
- Author – somebody who can publish and manage their own posts.
- Contributor – somebody who can write and manage their own posts but cannot publish them.
- Subscriber – somebody who can only manage their profile.

Therefore, by targeting such a forum a wider range of users who do more complex tasks and thus have a more mature level of technical competencies are targeted.

Finally, SAP User Groups provide a valuable channel through which SAP gathers feedback concerning the problems and requirements of its users in all technical and functional areas

of interest. This will give us a view of users who use tailored softwares in different industrial areas, such as Retail, Finance, or Human Resources.

Using software in the data analysis process has been believed to increase consistency and/or accuracy of qualitative research (Lu and Shulman 2008). Software tools provide a degree of accessibility and efficiency, enhancing the overall level of organisation of any qualitative project. Researchers enhance their ability to examine, sort, filter, search, and think through the identifiable patterns as well as peculiarities in large datasets. In our research, NVivo 10 (International Copyright © 1999-2014) was used in the data collection and analysis.

Moreover, multi-coder arrangement (Crawford, Leybourne et al. 2000) is valuable to reduce subjectivity and bias. Research which uses a single coder to mark themes relies on the coder's ability to accurately and consistently recognize examples. Having multiple coders analysing the text increases the chance of finding all the examples in a text that relate to a given theme. It helps increase the reliability of the analysis process. Furthermore, multi-coder arrangement can also serve as an external validity assess for the coded data, demonstrating that multiple coders can select the same text as relating to a theme. This helps validate that a theme is not just emerging from a single coder subjective thinking.

Therefore, two researchers performed the coding and analysis of forums. The first researcher is the PhD student, who is the main member in this research. The second research participant was engaged to ensure the above benefits are met. This participant works as a senior software engineer in an International Software house based in Germany. He has special expertise in engineering enterprise resource planning systems using SAP, and online stores using Hybris. Also, he has experience in dealing with customer problems through feedback loops, solution proposals, requirements verification, and software acceptance. Therefore, both researchers have common knowledge ground, which will lead to more fruitful discussions.

Both researchers coded the same collection of sources. To start with, the two researchers both sat and reviewed the initial stages of coding to ensure they have similar interpretation of the codes, and they started with the same initial template in Figure 8. A node hierarchy was created for each team member with a definition for each node. Nodes can be themes, concepts, categories and/or codes. After each team discussion, the members refined, merged and/or reorganized the nodes. Thus, credibility of the findings was ensured and maximized by the inter-coder agreement and academic advisor's auditing (Miles and Huberman 1994, Creswell 2012, Creswell 2013).

4.4 Forums Analysis Study Results

As a result of the focus groups analysis, the final thematic map in Figure 7 shows broad results of users' feedback aspects (Sherief, Abdelmoez et al. 2015). Each thematic area contributes to the body of knowledge from a different angle. Thematic areas can be viewed from two different perspectives. In the first perspective, participants gave several insights regarding the structure of the feedback and what are the characteristics they think makes their feedback meaningful and useful. These ideas are covered in the environmental and structure thematic areas. In the second perspective, participants gave their perceptions regarding what they expect from a feedback tool. How it can support, motivate and value their feedback. These ideas are covered in the engagement and involvement thematic areas.

Users' Feedback can be and has been the driving force in software evolution. The stakeholders of software include users who utilize the software to reach their needs and expectations, i.e. requirements. Thus, users' acceptance and efficient use of the software is a main goal in software development and evolution. In a dynamic world, users' acceptance and view of the software would be also dynamic and would need to be captured throughout the life time of the software to stay up-to-date.

4.4.1 Initial Template

The template shown in Figure 8 is the initial template for the forums analysis. It is derived from the final thematic map of the focus groups shown in Figure 7 with a focus on the environment and Structure thematic areas. This template was edited and enhanced in the forums analysis process by each researcher. In Section 4.4.2 to 4.4.5 the final results reached during the analysis process to the same data sources is explained, and the final thematic map of forums analysis shown in Figure 9 (Sherief, Abdelmoez et al. 2015). Intermediate results of the forums analysis are shown in Appendix 3.

The thematic mapping analysis technique was used to analyse the forums (Braun and Clarke 2006). ***Inductive analysis*** coding was employed, which is coding the data without trying to fit it into a pre-existing coding frame, or an analytic preconceptions. In this sense, this form of thematic analysis is data-driven.

To start with both researchers created the same node hierarchy on NVivo 10, with a clear description for each node to ensure a common understanding of the nodes meanings and essence. The coding process started with the TechNet forums. It was chosen to start with, because users on TechNet have least technical experience compared to the other two other targeted forums (i.e. WordPress and SAP), and users with minimum technical experience

tend to elaborate more in their feedbacks. This has led to many enhancements in our node structure.

During the coding process five team discussions were held that led to changes to the initial node structure in the initial template until the final thematic map was. After each discussion refinements were made and the node structure was unified, and reorganised to fit in the new structure. In this section the sequence of intermediate changes made will be described.

Several types of changes were made in the initial thematic map and the intermediate maps, ranging from adding, removing, moving, and renaming themes and/or codes. These enhancements and/or changes were deemed necessary after discussions on evolving thoughts related to what both researchers examined in the forums. The intermediate results are detailed in Appendix 3.

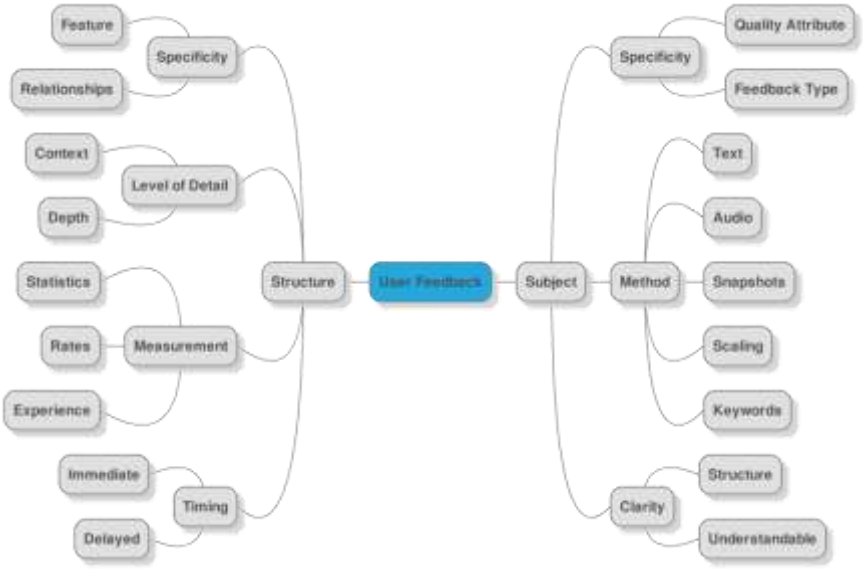

Figure 8. The Forums' Analysis Study Initial Template

4.4.2 Feedback Type Thematic Area

The first thematic area that was founded from our analysis to the forums is the **novel classification of feedback types** that users provide. Ten distinct feedback types that users use on forums were reached. In this section definitions for each type of feedback with exemplification will be provided. Also, some patterns of use for each feedback type will be concluded. These definitions and observations will serve as a strong base for our next step in the research, which is building ontology of feedback structure concepts. These definitions will be translated into rules that uniquely identify each type of feedback and serve in controlling how users can provide richer, more meaningful and useful feedback.

Before starting to define the meaning of each feedback type, there are two different kinds of feedback, a ***simple feedback***, and a ***complex feedback***. A simple feedback is a feedback that consists of a single feedback type that a user provides in his post to express a certain meaning, while the complex feedback is a structured feedback that consists of several feedback types that together form a new meaning that can be inferred from its unique structure. Below if the list of feedback types and subtypes (i.e. cases):

Figure 9. The Forums' Analysis Study Final Thematic map (Sherief, Abdelmoez el al. 2014)

4.4.2.1 Confirmation/Negation

A Confirmation or Negation is a ***simple*** feedback type that the users use to agree or disagree on problems or opinions of other users. When these feedback types are unaccompanied with other types in a feedback, it can be inferred as voting for a problem or a given solution.

For example, if user A posted a problem and user B has the same problem, user B may just agree on what user A said (i.e. instead of writing a whole feedback with the same meaning, the user can reference a feedback and adds his confirmation or negation). This adds an extra vote for the problem. This may be useful to prioritize problems by knowing how many users are having this same problem. For example: *"I have the same problem"*, or *"We are facing the same situation described in the original post. The Team Calendar for the approval process is blank unless there is a pending request to be approved. All other calendars work correctly."*

Also, they can be used to rate whether a solution/suggestion was able to solve a problem or not. For example: *"Made absolutely no difference. With all due respect, I think you only skimmed over my post and offered a generic response."*

Observation 1:

From the analysis it can be concluded that: a) Confirmations and Negations can be used in isolation to rate a problem, or vote for a solution; b) They are two disjoint feedback types which means that cannot be used together in the same feedback; c) They can be used in association with other types in the same feedback to convey a certain meaning or infer a new feedback type.

4.4.2.2 Investigation

An Investigation is a ***simple*** feedback type used when a user is asking a question to clarify something about another feedback posted by another user. A user may ask about some issues in a problem statement, or unclear steps in a provided solution, or clarify some contextual information that helps explain the problem more.

For example: *"Can you tell me why the (Remove personal information...) option is greyed out?"*, *"Which version of Word are you using?"*

Another example: "Please tell us how you migrated BPC 10 to 10.1? Have you activated environment shell in BPC 10.1 before the migration?"

Observation 2:

From the analysis it can be concluded that: Investigation are not used to clarify Confirmation, or Negations, because mostly these feedback types consists of simple agreements or disagreements that do not need clarifications.

4.4.2.3 Elaboration

An Elaboration is a ***simple*** feedback type where the user gives extra explanation on a feedback he already posted. There are two cases for giving extra explanations on a feedback.

4.4.2.3.1 Feedback Elaboration

First case is when a user needs to give more detailed information that he forgot to provide in his main feedback this can be added separately in the feedback where he elaborates. For Example, A user can elaborate on a problem he provided by giving explanation on some trials that he made trying to solve his problem or rephrasing the problem statement.

For example: *"***[Rephrasing the same problem**] *I am not able to get data during run of RSA3 in source system. I am not sure why update mode is F and not D. It is disabled to change during entering DS name at RSA3 in source system.* **[Adding explanations of trials that were made in attempt to resolve the problem]** *I have tried init without data transfer and did not get any issue during run of the IP and also I can see delta queue created in RSA7. When I tried the second Info package i.e. delta update, I got the following error"*

Another case, if a user elaborated more in a mitigation he provided by added some information about how this Mitigation has to be applied. For example: *"Note that the (Remove personal information...) option gets selected after you have run the Document Inspector on a document. You have to clear the option manually, or it will be sticky (as you have noticed)."*

Observation 3:

From the analysis it can be concluded that: a) Feedback Elaboration can reference the types of feedbacks where more explanations may be needed such as Problems, Mitigation, or Verification; b) Feedback Elaborations do not reference the feedbacks types where extra explanation is uncommon or not needed such as Confirmations, Negations, Investigations when they are used in isolation (i.e. unaccompanied with other feedback types).

4.4.2.3.2 Investigation Elaboration

Second case is when a user simply replies on an Investigation by giving detailed explanations to answer the posted question(s).

For example: *"***[Investigation]** *Please tell us how did you "migrate" BPC 10 to 10.1? Have you activated environment shell in BPC 10.1 before the migration?* **[Answer]** *Yes, the shell has been activated. Backed up 10.0 ENV using UJBR and restored it in 10.1. It is not working."*

Observation 4:

It can be concluded that the Feedback Type Investigation Elaboration can only reference a Feedback Type Investigation as this type acts as an answer to question type.

4.4.2.4 Justification

A justification is a ***simple*** feedback type used when users need to provide reasons to support their feedback. They may give reasons why they provided a solution/ suggestion, or it can be used with confirmations or negations to state reasons why a user agrees or disagrees on a feedback opinion of another user.

For Example: *"Because the (Remove personal information...) option is greyed out with the Document Inspector, this was the trick that fixed it for me."*

Another Example: *"No; I can't repair PowerPivot standalone, because it is embedded with Excel. It's strange, if you used 64bit office, there seems no 64bit odbc drive now. You'd better switch to 32bit office."*

Observation 5:

From the analysis it can be concluded that: a) Justifications and Elaborations are two disjoint Feedback Types that cannot be used in the same feedback; b) Justification cannot also reference Investigation feedback types, because users do not tend to give reasons why they are asking a clarification question.

4.4.2.5 Verification

Verification is a ***complex*** feedback type where a user gives his opinion on a solution or suggestion he received on the problem that he posted. As a complex type it means that it combines several other feedback types in its structure that are mandatory in its definition. Specifically in order to verify whether a solution or a suggestion was useful or not, this

feedback has to reference a certain Mitigation (i.e. Solution or Suggestion) in which the user will be giving his opinion to verify whether it solved the issue or not by using Confirmation or Negation.

For example: *"Thanks Ed. That's the type of thing I was thinking. An alternative method (really just a rephrasing of your concept) would be to generate various powers of the matrix using mmult, and then to iteratively create higher powers of the matrix."*

Observation 6:

From the analysis it can be concluded that: a) Verification must reference a Mitigation (i.e. Solution or Suggestion) Feedback Type; b) has to provide a Feedback Type Confirmation or Negation first to show whether the Verification is an agreement or disagreement on the provided Mitigation; c) it is desirable that users provide Justifications on why they are verifying the solution.

4.4.2.6 Problem

This feedback type refers to a certain feature or group of features in the software that the user is having problem with, and a detailed explanation of the problem. Problems may use other feedback types such as Investigations to ask users some questions they need answers for.

However, problems in general cannot occur in the same Feedback post with Mitigations or Verifications. In general users who post problems are not the same users who post the Mitigations, and even if this case occurred will not be contained in the same problem post. Same for Verifications on Mitigations will never occur while posting a problem as there is no Mitigation yet to be verified. Moreover, a problem post may not come as a reply.

Observation 7:

From the analysis it can be concluded that: a) problems must specify feature(s) in their definition; b) must provide detailed explanation to increase understand ability; c) can be accompanied with other Feedback types in the same feedback except Mitigations and Verifications.

4.4.2.6.1 Topic Definition

Topic definition is a ***simple*** feedback type that represents the first posted problem in a feedback thread where the user is seeking help. Therefore it does not reference any other feedback in the thread but can be referenced in many other posts.

For example: *"***[Feature Specified]** *When I use the Track Changes feature in Word 2013*

(running on Windows 8.1), **[Detailed Explanation]** *and I create a new Comment, my name temporarily appears as the Author, but when I close and re-open the document, it switches the Author name to "Author" instead of my name. Also, my picture disappears in favour of a generic icon.* **[Investigation]** *What is going on here?* **[Context Information]** *FYI, I'm logged into Windows 8.1 using my Microsoft Account. I'm not sure if that matters. It seems to be pulling my name and account picture correctly, until I close and re-open the document*."

4.4.2.6.2 Addition

This is a ***complex*** feedback type where a user votes (i.e. agrees or disagrees) on any posted problem, and adds another problem in his feedback, which is not related to the main problem on which the discussion is held. This means that a feedback thread may contain multiple problems along with the replies.

From the definition of this feedback type as a complex type, this implies that it must contain other feedback types in its definition, which in this case are Confirmation or Negations that must reference another problem. Therefore, it cannot reference a feedback post that contains Mitigation, because by definition this feedback is used to add a problem to a problem.

Observation 8:

From the analysis it can be concluded that: a) an Addition requires a Confirmation or Negation first that references a problem; b) it must include the definition of a new problem; c) it cannot reference (i.e. come as a response to) a Mitigation Feedback type.

For example: *"***[Confirmation on a Problem]** *I'm having a similar problem*. **[Addition of a new Problem and explaining it]** *For this particular Word file (Word 2013), my name and time stamp disappear on balloon comments when I reopened it, although new comments had my name and time stamp. (Of course I don't know what will happen to those new comments!) Not only this, but when I used the Compare feature to another document, it would only come in Draft mode. I was unable to unclick the Draft icon*. **[Investigation]** *Any ideas? Thanks in advance!"*

4.4.2.6.3 Extension

Is a feedback type that references an existing problem and tries to add an extension to it. The new introduced problem is related to the existing problem that it references. Unlike the Addition Feedback Type that adds a new unrelated problem definition to the feedback thread.

All problem types: Topic Definition, Addition, and Extension are disjoint types, which means that they cannot occur together in the same feedback post (i.e. the user **cannot** add a new Topic and at the same post confirm/negate it and add an unrelated problem, or extend it). There are two cases for an Extension that is discussed in the next sections.

Observation 9:

From the analysis it can be concluded that: All Problem Types are Disjoint. The main type is Topic Definition, and the other types occur as a result of discussions.

4.4.2.6.3.1 Problem Extension

This is a ***complex*** feedback type where a user tried a Mitigation and it solved part of the problem, BUT led to another related problem to occur. As a complex type this implies some restrictions to be put into consideration. In this case the user tried a mitigation that solved part of his problem, which means that the user is Confirming that he is having the same problem as the user who posted it (i.e. user rating the problem), and also he tried a solution or suggestion that were posted to try resolve the problem and adds a Confirmation (i.e. Agrees) to that Mitigation. However, after trying this mitigation new problems evolved.

Observation 10:

From the analysis it can be concluded that: a) Problem Extensions contain Confirmations that reference a Problem type; b) Problem Extensions contain Confirmations that reference Mitigation; c) Problem Extensions introduces new problems (i.e. the extension that is still unresolved by the Mitigation that was applied).

For Example: *"***[Confirmation on existing Problem]** *Our office has been struggling with a related problem that maybe you can solve.* **[Explanation of the new Extended (related) Problem]** *Basically, the same person is repeatedly given a different reviewer name as they work in a document (presumably every time the document is auto saved). For example, if I work for an hour adding edits or comments on a document by the time I'm ready to share it will look like five different people made changes.* **[Confirmation on Mitigation that solved part of the problem]** *The Inspect Document fix works great to remove all the extra reviewer names, but it changes them all to 'Author'".*

Another Example: *"***[Confirmation on Mitigation that solved part of the problem]** *Thank you! Good news and bad. I found the reference to eane in the cert manager, and removed it. I had been in there before I knew I was looking for eane, so I should have realized to go back and search again.* **[Explanation of the new Extended (related) Problem]** *Anyway, the user gets into outlook and can send/receive ok but when launching outlook, he gets a*

variation of the same security alert. This time, instead of autodiscover.ourdomain.com it says ourexchangeservername.ourdomain… **[Confirmation on existing Problem]** *I see we have the chance to install the cert but I don't know where to install to..."*

4.4.2.6.3.2 Mitigation Trial Failure

This is a ***complex*** feedback type, where a user confirms on a posted problem (i.e. he has the same problem), AND tried the mitigation that was posted by other users in attempt to resolve the problem, BUT couldn't try the mitigation (so this is a new problem for him besides the main one). As a complex type this implies some restrictions to be put into consideration. Since the user has the same problem, therefore there is a Confirmation on a posted problem that can have any of the problem types such as Addition, or Topic Definition. Moreover, the user attempted to try a Mitigation, which he couldn't apply. Therefore, he uses a Negation to reference the Mitigation he could not apply.

Observation 11:

From the analysis it can be concluded that: a) Mitigation Trial Failures contain Confirmations that reference a Problem type; b) Mitigation Trial Failures contain Negations that reference Mitigation; c) Mitigation Trial Failures introduces new problems (i.e. the extension that is still unresolved by the Mitigation, because the Mitigation couldn't be applied).

For Example: *"***[Confirmation on a Problem in the previous posts]** *I have the same problem here!* [Negation on a suggested Mitigation that the user couldn't try] *Can you tell me why the "Remove personal information..." option is greyed out and how it could be made active? I tried using the document inspector but to no avail.* [The new extended problem is how to try the Mitigation] *How do I ungrey it?"*

4.4.2.7 Mitigation

Mitigation is a ***complex*** feedback type that represents a solution or a suggestion that may help a user resolve the problem(s) he has. Since this type is intended to resolve a problem, therefore it has to reference that problem in the solution or suggestion for specificity. Also, for every Mitigation it is always expected that the user who posted the problem will Verify that Mitigation. There are two types of Mitigations:

4.4.2.7.1 Solution

A solution is a well-known procedure or steps that when followed can resolve the problem or issue. For Example: *"Do the following to start the Document Inspector in Word 2013: Click File | Info | Check for Issues | Inspect Document. In the list of content, make sure that "Document Properties and Personal Information" is checked and then click the Inspect*

button. Click Remove All (next to the "Document Properties..." item). Save, close and reopen the document. From then on, your user name will be replaced with "Author" each time you reopen the document."

4.4.2.7.2 Suggestion

A suggestion is a recommendation that a user provides for another user as a trial to resolve his problem. This suggestion may or may not solve the problem. This needs Verification from the problem owner (i.e. the user who posted the problem).

For example: "**[The user tries to explain more about the feature]** *OK - the idea of find disputes it to filter by the selection parameters (you can define these) the idea of "my disputes" is to display where your user ID is the processor, co- coordinator or person responsible.* **[The user starts to use conditions and suggestion trial for each condition, which indicates that the user is not sure whether this will solve the problem or not or he does not understand the problem precisely]** *If you are working in a dev system - you might assign all disputes to yourself and therefore you will be the processor for all disputes. If you amend the processor to someone else, and then try and find it in the my dispute - it should not be there (if you are sorting by processor) If it still appears - you have an error with your build"*

Observation 12:

From the analysis it can be concluded that: a) Mitigation should reference problems they are trying to resolve; b) There are two types of Mitigations, Suggestions and Solutions, which both differ in the way users describe their statements and also on whether users are precise or not.

4.4.2.8 Correction

A correction is used is used when a user corrects the understanding of another user. There are two cases for this feedback type.

4.4.2.8.1 Problem Correction

Problem correction is a ***complex*** feedback type. It occurs when the user corrects the problem of another user. In a problem definition a user must refer to a feature(s) that he is having a problem with. Sometimes the user is using a feature which is not intended for the type of task he is doing, simply due to a lack of understanding of the job a feature should perform. Consequently, other users can provide corrections to this misunderstanding.

For Example:

*"***[Problem]** *is it possible to change the templates that are shown by default when a user clicks 'File', 'New'? My organization would like the default template to have a corporate header and footer by default on new docs."*

*"***[Problem Correction starts with Negation on the problem statement made above]** *the default template is the normal template; it should not have headers or footers.* **[Problem Correction Explained]** *However, you can set Word up to use a different template for new documents and upon start-up. Download the examples on my Add-Ins section. Specifically, look at the Letterhead Add-Ins and the Easy New Document Template package."*

Another Example:

*"***[Problem]** *I need to display all Dispute Cases via SCASE --> Find Dispute Case Option. However, when setting up UDM_SP_CASE_LOCATOR I can see that both UDM_SPS_CASE_LOCATOR and UDM_SPS_MY_CASES have connection parameter values "MY_CASES_ONLY"."*

*"***[Problem Correction starts with Negation on the problem statement made above]** *Sorry - you are wrong.* **[Problem Correction Explained]***It is possible to use UDM_DISPUTE to display all disputes cases and check its progress. That would be done via the "find dispute" option. If you don't want them to - that is a business reason and not a functional constraint."*

Observation 13:

From the analysis it can be concluded that: a) Problem Corrections must contain some Negation that references a Feedback Type Problem statement; b) Problem Corrections adds explanation for feature usages, and increases users' awareness of unknown features; c) Problem Corrections and Mitigations are disjoint Feedback types that do not occur in the feedback.

4.4.2.8.2 Mitigation Correction

Mitigation Correction is a ***complex*** feedback type. This type of feedback may occur when a user is trying to correct a Mitigation that was provided for a certain problem. Errors in Mitigations may occur due to the lack of contextual information about the tasks the user is doing or environmental information about the softwares or hardware used while applying Mitigation.

For example:

*"***[Suggestion]** *That page shows this at the bottom:*

<! -- Dynamic page generated in 0.754 seconds. -->

<! -- Cached page generated by WP-Super-Cache on 2013-09-29 11:36:32 -->

Try clearing the Super Cache plugin."

*"***[Mitigation Correction]** *I use W3 Total Cache not Super Cache,* **[Mitigation Correction contains disagreement (i.e. Negations) on Solutions/Suggestions]** *but in any case I've both cleared all caches and the page cache several times with no effect."*

Observation 14:

From the analysis it can be concluded that: a) Mitigation Corrections must contain some Negation; b) Mitigation Corrections must reference a Feedback Type Solution or Suggestion; b) Mitigation Correction, Problem Corrections, and Problems are disjoint feedback Types that do not occur in the same feedback

4.4.3 Level of Detail Thematic Area

The second thematic area that was reached from the forums analysis is the Level of Detail. Level of Detail represents how much information the user provides in their feedback to express their opinions or problems. The information users provide have two major categories: Depth and Context. A single feedback can contain a mix of contextual information and several kinds of Depths.

Depth means how detailed the user is in expressing their feedbacks. There are seven **novel categories of detail types** the user can use while providing his feedback. However, some patterns were concluded for: use of these categories, as their usage differs according to the feedback type the user is providing.

Context means the information the user may provide about the settings of his use to the software or while providing his feedback, which may affect the problems, mitigations, other users' responses. Therefore, this thematic area is considered a complementary area to the feedback types explained in section 4.4.2, as it adds more clarity to the feedback descriptions.

4.4.3.1 Depth

4.4.3.1.1 Concise

The first category of Depth is Concise. By literal meaning it is used when users provide very short feedback types with no explanations or details. From the analysis it was noticed that it is used mostly, when users tend to confirm or negate by just expressing their agreement or disagreement on a feedback. Moreover, it was never used in problem statements or mitigations, since by nature these specific feedback types need explanation to be meaningful.

Observation 19:

From the analysis it can be concluded that: a) Concise is a short description for a feedback type, which can be used with Confirmations, Negations, or Verifications; b) Concise can never be used to detail a Problem, Mitigation, Elaboration where users are expected to provide more details and be clear enough.

For Example: *"***[Negation that is used to Verify Mitigation as unable to resolve the problem]** *I'm having a similar problem and the above solution didn't work."*

Another Example: *"***[Confirmation that is used to Verify Mitigation as able to resolve the problem]** *Thanks to you! I was able to find the flag and correct the problem."*

4.4.3.1.2 Explanation

Explanation is the opposite of concise, as in this depth category the user is expected to provide as much details in his feedback to make it meaningful for other users. There is no restriction on the use of this depth category with any feedback type, because it is always acceptable to give more details especially in forums. However and as explained in the Concise section, it is obligatory to use explanations when explaining Problems, Mitigations, or Elaborations.

For Example:

*"***[Problem]** *we've got one end-user out of 65 getting a certificate error when starting outlook 2010. He launches it, gets the security alert error saying "autodiscover.ourdomain.com Information you exchange with this site cannot be viewed or changed by others. However, there is a problem with the site's security certificate". Any ideas would be appreciated! Thanks!"*

4.4.3.1.3 Exemplification

Exemplification is a Depth category that is utilized when the users need to provide examples within this text. In the forums' threads that were analysed examples are always given within explanations especially problem explanations.

For example: *"…and when I change my VBA to skip the element if that cell is empty, the resulting XML shows errors in VS2013 that the data is incomplete.* **[Exemplification]** *For example, VS2013 highlights that* ArrearsStartDate *is missing even though I can clearly see the below in the XSD file (crazy!)."*

Another Example: *"We force Excel to close, and the source Excel file is gone! We find there are two files that were created. One with a .tmp file extension, the other has no file extension.* **[Exemplification]** *For example, in one case the files were named 3F04D520 and 31545502.tmp."*

4.4.3.1.4 Trials

Trials is a Depth category used closely with problem description where the problem owner who is explaining the problem, shows that he made many attempts to resolve the problem but have failed to reach a Solution. The user posts these trials as a kind of extra explanation of the problem and how it occurs, and also to avoid getting suggestions from other users with same trials that he already made.

For Example: *"Researching on the web, I found a few things to try such as deleting the certificates key in the registry (Yes I exported it first), adding the line 127.0.0.1 localhost to the hosts file."*

Another Example: *"I am using Microsoft Office Pro Plus 2013 on Windows 7, 64-bit version. I have tried reinstalling Microsoft SQL Compact Edition 2005 and reinstalling Office 2013. I have run out of ideas and am a loss what to do next?"*

4.4.3.1.5 Scenario

Scenario is a Depth category which the user uses to explain text in a list. A solution can be explained in steps. These steps if verified by the problem owner can be used as a solution scenario to solve similar problems to other users. Moreover, other users may list the problems they have in the problems statement. Other may suggest mitigation to other users in a form of a list of possible actions to try; sometimes it matters to be in a certain order.

For example: *"***[Suggestion]** *Hi, 1. Please check the postings are done are not 2. Check whether they struck Ed in Inbound or Outbound. 3. Check in SM58 any TRFC's"*

Another Example: *"***[Problem]** *I am doing environment migration from BPC 10.0 NW to BPC 10.1 (Classic). Migration was successful, but I have the following issues: 1. Migrated BPF templates are not accessible in 10.1. It is not allowing me to create new template as well. 2. Not able to open migrated reports in 10.1 - Nothing is coming out. 3. Migrated logic scripts are not getting copied to 10.1 Classis - they are empty in 10.1(I can still manage this by manually copying script from old one!) Pls. help to fix the above."*

A Further Example: *"***[Solution]** *1. Please go to Review Tab, 2. Click the icon in right-bottom of tracking section, 3. Check and tick the box of "Pictures by Comments" to solve the problem."*

4.4.3.1.6 Feature Definition

This is a depth category which the users use to define their perception of the usage of a certain feature. This description is sometimes used in problem statements, which helps other users understand why the user is having a problem (i.e. sometimes users have wrong understanding of the usages of a feature). Moreover, users who provide Mitigation may use it a form to document how they use a feature with certain types of tasks. Finally, it is mostly used when users provide Feedback Type: Correction, specifically Problem Correction, where the user corrects the misunderstanding of another user by providing the correct feature definitions to features referenced in the problem statement.

For example: "*However, when setting up UDM_SP_CASE_LOCATOR I can see that both UDM_SPS_CASE_LOCATOR and UDM_SPS_MY_CASES have connection parameter values "MY_CASES_ONLY".* **[Feature Definition]** *I think this is what restricting display of all dispute cases is. Users can only see what is assigned to them.*

Another example: *"[***Feature Definition]** *Excel can perform matrix multiplication (e.g. A1:B2 times A3:B4) by the mmult function"*

Further example: *"***[Feature Definition]** *MBOX is a standard plain text file format for storing email messages on hard drive. In my opinion, there is no manual method for transferring entire emails of Apple Mail into MS Outlook PST format"*

Observation 20:

From the analysis it can be concluded that: Feature Definitions use with Problem Corrections is mandatory. By definition of problem correction, it is only possible when a user doubts the understanding of the problem owner's perception of a feature usage. There is no restriction on using feature definitions with other feedback types.

4.4.3.1.7 Question

Question is a simple depth category that is used with Investigations to indicate the question(s) posted for clarification.

For example: *"Based on your description, when your users tried to save the excel file, it crashed and even lost this file.* **[Investigations]** *Could you tell me where did you open and save the excel file? From local computer or network? If they open the same Excel file or random files?"*

It can be used sometimes in problem statements where the user asks for help, but this does not add any valuable details to the feedback thread, just a way for the user to confirm that he needs help and waiting for suggestions. On the other hand, it cannot be used with Mitigations.

Observation 21:

From the analysis it can be concluded that: a) Questions is a short description textual Depth Category used for Investigations; b) Questions can never be used to detail a Mitigation, Elaboration, Corrections, or Justifications where users are expected to provide more details and be clear enough.

4.4.3.2 Context

Contextual information can carry valuable information that can help make the feedback more understandable or useful. There are five main categories of contextual information that were captured in the forums analysis that map to (Krogstie, Lyytinen et al. 2004).

4.4.3.2.1 Task

It captures what the user is doing. This is specifically important when the user is describing a Problem feedback type, because it gives to the other users an idea about the context in which the problem occurred, or describing the frequent jobs that the user is involved in in his daily work which helps give an idea to other users about the importance the feature the user is having problem with.

For example: *"***[Task: the user describes frequent task in his job to add importance]** *I work on long documents over many days and need to be able to make and edits in a continuous track session."*

OR *"***[Task: the user describes frequent task in his job to add importance]** *I am in Outlook all day long and any steps that can save me 30 seconds of having to click to a different screen and back again, adds up very fast"*

Another Example: *"***[Task: the user describes what he was doing when the problem occurred]** *I am attempting to replace the older iView (leave request approver and team calendar) with the newer iView (approve leave request) which seems to combine the two functions.* **[Problem]** *When implementing the approve leave request iView, I get the following message "No Team setup for the user in the selection period. Contact Administrator."*

4.4.3.2.2 Spatio-Temporal

In this kind of context the user specifies information related to place and time. From our forums analysis an angle was found where such information may play useful role. Cases are when users try to explain the timing relationship between two tasks (i.e. two tasks happening together, or one feature corrupts when a user does a certain action). For Example: *"When I close the document then open it and change text that I had inserted and tracked in my last session, it shows the change as mark-up on mark- up."*

Another Case is when users try to specify some information about a problem in relation to where it occurs in software for example in a certain interface, or when using a certain module. For example: *"I recently moved my site to a new host and* [Spatio Temporal] *now the Media Library only shows a couple dozen images (out of hundreds). The images appear on the posts and pages, but not in the library."* A Further Example: *"It was so long time ago I designed the site. I would never have found it. Big thanks Very hard to edit I must say. You might wonder why no edit button exit, taking you to widgets in the normal interface."*

4.4.3.2.3 Personal

In this kind of context users express their emotional judgments, stress, or information about their expertise, which is repeated mainly with Negation feedbacks.

For example: *"***[Personal]** *I am an unhappy outlook user, as an IT professional who has to support over 300 people, I am making it very clear to my staff and co-workers that this is a limitation with no acceptable solution"*

Another example: *"***[Personal User expertise]** *I'm not good at cert issues because they don't come up very often", and "This is really frustrating".* Or *"This is absolutely unconscionable. I have been in this business of software development for 40+ years and never seen anything so rigidly pretentious."*

4.4.3.2.4 Social

Social means context information related to a user's role at work, information about co-workers…etc. For example: *"I work for different bosses. One boss on Monday, Wednesday and Thursday. I need the out of office option for all other days. So on Saturday, Sunday,*

Tuesday and Friday I need to have the OUT OF OFFICE setting in Outlook and NO I do not want to configure a rule for this"

Another example: *"We've got one end-user out of 65 getting a certificate error when starting outlook 2010."*

A further example: *"***[Social]** *I have seen this with 4 users in the last day.* **[Task]** *They are working and saving occasionally. At some point when they save, Excel stops responding."*

4.4.3.2.5 Environmental

Environmental context means context information related to a software or hardware specs, versions, architectures…etc. Users can provide these kinds of information in a problem statement to specify the software version they are using which may differ in the feature with problem from older or newer ones. Therefore, this adds specificity and usefulness to add such information. Moreover, users can add also environmental context in Mitigations to specify that the suggestion or solution works on a certain version, or works well with a certain hardware configuration.

For Example: *"Environment is exchange 2010 with outlook 2010, by the way."*, or *"When I use the Track Changes feature in Word 2013 (running on Windows 8.1)"*

Another Example: *"We are also experiencing the "EXCEL.EXE version 14.0.7151.5001 stopped interacting with Windows and was closed" error on Windows 7 SP1 32bit, but the behaviour is a bit different for the user."*

4.4.4 Method Thematic Area

In the forums analysis, it was noted that users use four different methods to provide feedback, which are: text, code snippets, snapshots and links. It was notable that some methods were frequently associated with a certain feedback types. The text method is the most commonly used method in all feedbacks, and even it is used with other methods such as links or snapshots. However, it is important to note that most users use text written in natural language, which leads to lots of misinterpretations. This motivates our goal in creating a new feedback modelling language that utilizes the same methods the users are used to provide their feedbacks with, but in a patterned way and with the aid of textual keywords. Therefore, this thematic area is considered a complementary area to the feedback types explained in the section 4.4.2, as it adds more expressiveness to the feedback descriptions.

Example of a textual feedback: *"Hi, Installing objects form content is same for any module. If you know the required objects then go to BW, RSA1--> BI content, from middle pane, find your object, grouping options as only necessary, drag the selected objects into right side*

pane. Choose install in background. There are more documents about bi content installations, please search and help yourself. Thanks"

Code snippets shown in Figure 10 are used to show fragments of code that have problems, or fragments of code to illustrate mitigation, and same for Snapshots shown in Figure 11. For example:

```
Function PowerMatrix(rngInp As Range, lngPow As Long) As Variant
Dim i As Long
PowerMatrix = rngInp
If lngPow > 1 Then
  For i = 2 To lngPow
    PowerMatrix = Application.WorksheetFunction.MMult(rngInp, PowerMatrix)
  Next
End If
End Function

Use it in a range like;

=PowerMatrix(A1:B2,3)
```

Figure 10. A Sample Feedback described by Code Snippet

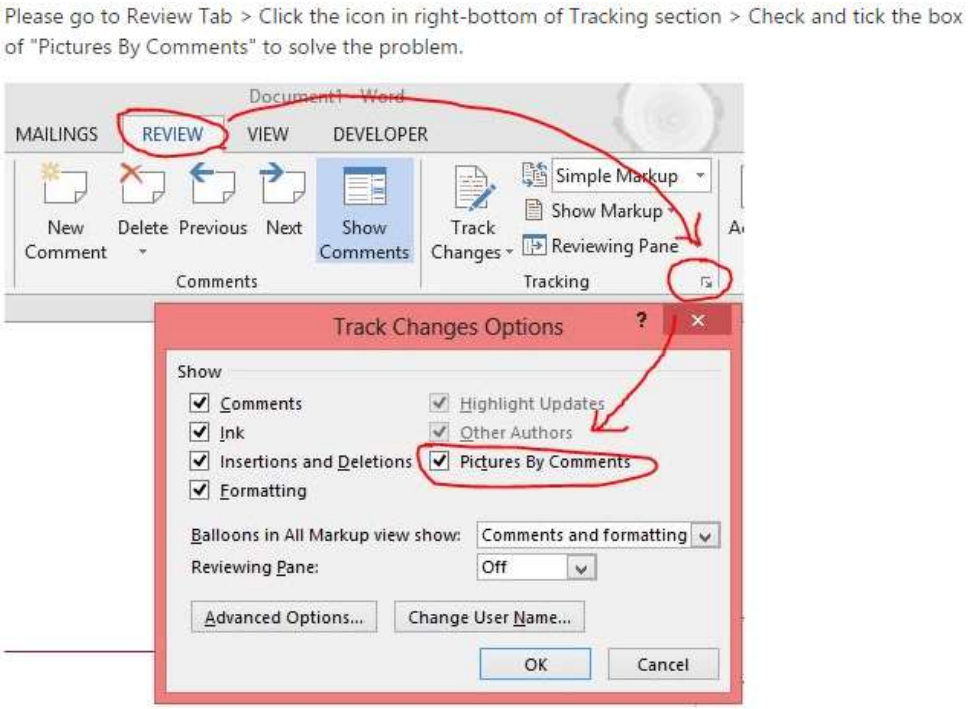

Figure 11. A Sample Feedback described by Snapshot

Finally, a further method used by users to express details in forums, is Links. Links are very useful in providing Mitigations whether solutions or suggestions. Users use them to provide all the information they need by referencing the page that contains manuals or illustration the may help the problem owner. They can also, provide extra notes or explanations in their feedback besides the Link.

For example: *"Install the certificate to trusted root certification authorities as introduced in this: http://www.slipstick.com/outlook/security-certificate-not-from-trusted-certifying-authority/ -Jeff"*

4.4.5 Measurement Thematic Area

Finally, this fourth thematic area concluded derived from this study is measurement. Measurement means measuring problem occurrence frequency or voting for mitigations' usefulness. This can simply be done through confirmations and negations that reference Problems or Mitigation feedback Types as we have mentioned in section 4.4.2.

The measurement concept is a very useful premise when it comes to making use of all the feedbacks gathered from users. By gathering such relationships between different users' feedbacks, it will allow the system to a) prioritize the problems according to its rate of occurrence and number of users; b) Also, it may help in evaluating the overall quality of the system, as it will provide a quantitative value of the system resolved and unresolved problems, which will assist in the comparison, and decision- making; c) Finally, when the system arrives to a good Mitigation action, the feedback causing this Mitigation could be reused in similar cases.

4.4.5.1 Problem Rates

It was observed that at times users may reply on others' feedbacks by stating how much they agree or disagree with the problem, for example: *"Agreed - The contrast in office 2013 is horrendous…"*, or whether this problem has occurred to them before or not, for example: *"We are facing the same situation described in the original post."*

4.4.5.2 Mitigation Votes

At times users may reply on others' feedbacks by stating how much they agree or disagree on the helpfulness of a Solution or a Suggestion. For example: *"***[Confirmation on another user's Mitigation: adding a Vote]** *Rich has answered your question as well as can be done. This has always been the way Track Changes works.",* or *"***[Negation on another user's Mitigation: adding a Vote]** *In terms of work around …This isn't a real work around"*

4.5 Threats to validity

Although the principles in conducting qualitative methods approach were followed carefully, this study still has five main threats to validity:

1) In the focus groups study users were students, researchers, and engineers recruited from Egypt and UK, which might produce a population bias;

2) A common threat to validity in focus groups study is whether all the participants perceived the questions as intended. This issue was addressed by providing scripts which went through iterative revisions and modifications by two research members to ensure clarity;
3) While the analysis of forums was effective in identifying and describing concepts that construct users' feedback, it is possible that it did not identify all the important aspects and factors that can affect and influence their behaviour in this regard;
4) The number of analysed feedbacks from the three different forums (200 feedbacks) could be found medium considering that numerous number of threads available online, analysis was stopped when no new further concepts were distinguished, however a bigger number of feedbacks might produce new concepts that were not identified;
5) In the forums analysis, forums where business users provide feedback were targeted; future research would further investigate general purpose forums (e.g. products, social media) to come with common concepts and more general results.

4.6 Summary

In this chapter the first two studies that attempt to achieve the research aims and objectives are introduced. The adopted research methods were explained in details, and the results are illustrated and described. The main contribution from the forum's analysis study was the new classification and definitions of feedback types and their constituents. In the next chapter, another study will be explained that explores the engineers' perspective to the utilization of feedback to accomplish evolution tasks.

5. Exploring Feedback Utilization - Engineer View Point

This chapter explains the study conducted with engineers in order to explore and observe the problems caused by using ad-hoc feedback structures in the communication between end-users and engineers during the maintenance and support phase. The chapter explains the detailed purpose of the study, the research method that was followed, and the results reached. This study serves as a link between the previous results reached concerning the structuring of feedback, and the next step of the research that is providing an engineering approach for feedback acquisition and utilization.

5.1 Purpose of the study

Users' feedback serves as a communication method between engineers and users at during the maintenance and support phase, where users can provide relevant information to guide engineers in accomplishing several software change identification and evolution tasks starting from interpreting the users' feedback to understand the problem and trying to reproduce it followed by estimation, prioritization, and planning. However, engineers and developers spend considerable effort trying to collect useful information needed from users that can help them in accomplishing these tasks both successfully and in a timely manner.

This study purpose is to investigate what are the problems in the maintenance phase that are triggered by both the lack of information from end-users and miscommunication between both the end-users and the software engineers. Another purpose is also trying to find the associations between the problems, causes, and types of missing information that affects the tasks' achievement, delays it, and causes extensive communication effort between the two parties to reach their aim. This will also assist in gaining a deep understanding of how the instances of formalized feedbacks entered by the end-users can be utilized to resolve communication, and evolution tasks problems.

This builds upon what the results reached from previous studies, as the analysis and querying of classified structured feedback will provide the engineers with important knowledge that can inform the evolution process and help keep requirements information up-to-date. The goal is to design a novel workflow that combines both concrete and formal instances of inter-related feedback, and systemized formal steps that can help engineers in resolving problems they meet and supporting them in different decision-making situations they encounter in the maintenance phase.

5.2 Research Method

Direct feedback from interested and concerned individuals is fundamental to achieve this part of the research. The interviewing technique will be used to gather information from engineers (Berg 2004) (Lazar, Feng et al. 2010). There are three fundamental types of research interviews: structured, semi-structured and unstructured. Structured interviews are, fundamentally, verbally managed questionnaires, in which a list of prearranged questions is asked, with small or no adaptation and with no scope for follow-up questions that permit additional elaboration. On the contrary, unstructured interviews do not reveal any predetermined theories or ideas and are performed with little or no organisation. Such an interview may simply start with an opening question and will then progress based upon the initial response.

Semi-structured interviews (Lazar, Feng et al. 2010) consist of several key questions that help to define the areas to be explored, but also allows the interviewer or interviewee to diverge in order to follow an idea or response in more detail. This interview format was used in our study, as it provides participants with some guidance on what to talk about. The flexibility of this approach, particularly compared to structured interviews, also allows for the discovery or elaboration of information that is important to participants but may not have previously been thought of as relevant by the research team.

The ability to go deep is perhaps the strongest argument in favour of semi-structured interviewing. By asking questions on a wide range of concerns and giving the interviewees the freedom to provide detailed responses is one way to gather data that would be very hard to capture otherwise. Also, ambiguities can be clarified and incomplete answers followed up. However, there are also disadvantages for interviews, which are: 1) they can be very time-consuming: setting up, interviewing, transcribing, analysing, feedback, reporting; 2) they can be costly; 3) different interviewers may understand and transcribe interviews in different ways. For the transcription process, one researcher conducted the interviews on-site and made the transcription process, and another researcher revised the transcripts and made any needed updates to avoid any bias or misunderstanding.

5.3 Software Company

Participants were recruited from a company that has been firmly established in e-business for more than 20 years. They provide services and specialize in software solutions for industrial customers and public administration. They provide variety solutions such as:

- E-commerce systems: they create shop systems for customers that include the entire process, from supplier integration to sales channels. So customers can market their products successfully and provide better services.

- E-procurement: efficient catalogues are the basis for innovative e-procurement systems. They automate general procurement of C-class articles, services and complex product groups for customers and help their projects to succeed in the SAP environment. This makes the procurement processes simpler and more profitable.
- Public Administration: they provide simple and reliable tailor-made management programs for vastly complex data required by government agencies and administrators. From practical experience they understand the language of public administration and know the special requirements, rules and standards that concern when managing data.

Since founded it has produced more than 1200 success stories with a large variety of International customers. A few of the company types they work with include: industrial digitalization companies, universities, international airports. Also, companies that design and operate online shops, Internet-based customer acquisition, internet marketing, developing partner networks and complex, highly efficient product picking and distribution logistics. Moreover, they work with wholesale companies for connecting and fastening technology. In addition to its broad product portfolio, the company also offers innovative logistics systems for the automotive, industrial, railway and trades industries.

For this research all the above mentioned experience in the software domain makes it a very resourceful candidate to perform the study in. They have large experience and real-case problems that happen in the maintenance phase, especially when communicating with their customers. Also, since it will be explored how the miscommunication of customers' opinions and problems through feedback affects the maintenance / evolution tasks, so interviewing several roles that are involved in the evolution process is needed. 10 consented interviews were conducted in the Alexandria branch with participants from 4 different roles, which are: consultants, software engineers, team leaders, managers both program and technical.

5.4 Interview Process

An eight stage process was followed for the preparation and analysis of the interviews, which is explained below in detail (eVALUEd 2006):

5.4.1 Determine the purpose of the study and what information is required

Below in Table 7 is the main list of questions that were prepared for the interviews. Mainly, these questions were designed to understand the problem situations that the participants meet during the maintenance phase, the relationships between the problems (i.e. how the problems they face affect other tasks or evolves other problems), and the kinds of missing

information that affects the evolution process tasks completion, delays it, or leads it to failure.

The themes that to explore are:

1) What are the problems that software engineers meet when receiving feedbacks from customers both from acquisition perspective and interpretation problems?
2) How do these problems impact the tasks they perform in the evolution process such as the identification, estimation, impact analysis, planning, and solution design?
3) What are the problems that result from lack of organized and systematic means of communicating with end-users using feedback, and other proper inputs such as requirements models?

Table 7. Interview Questions mapped to the themes explored in the study

No.	Interview Question	The Theme it links to
1	What is your current role?	This question relates to theme 2, as it is important to know the role in order to know the tasks related to the role. And thus be able to know when and how they are involved in the maintenance phase. And when problems are discussed it could be related to tasks and/or roles.
2	How do customers request modification or report problems?	This question relates to theme 1, as participants are expected to provide explanations to the current methods of feedback acquisition and their problems.
3	What are problems that occur during the perception of customer requests? How do you handle these problems?	This question relates to theme 1 and 3, as participants are supposed to provide explanation of the current methods or tools used in feedback analysis and their drawbacks. Also, problems in the feedback content itself, and how to they communicate with end-users to acquire the inputs they need.
4	What are the problems that occur in the estimation tasks? What kind of information from users that can inform your task estimation decisions?	This question relates to theme 1 and 2, because it tries to capture information about the type of details that end-user feedback should contain and how the lack of this information affects and propagates other evolution process tasks.

5	What are the approaches you use in order to identify the impact of a change on the system?	This question relates to theme 2 and 3, because this question tries to find if the needed information that should be communicated from the end-users or existing RE models that could help the engineers better identify the impact of a change.
6	What are the customer-to-engineer communication problems that affect your design decisions and the way you approach the problem?	This question relates to theme 2 and 3, because this question tries to find if the needed information that should be communicated from the end-users that could help the engineer to design a solution and hand it over to the end-users for discussion and/or approval.
7	As a software engineer, from your observations what are the criteria that customers put into consideration while prioritizing their problems?	These questions relate to theme 1, 2, and 3, because these questions try to find how engineer expect end-users to communicate prioritization issues in their feedback. Also, if there are conflicts how are they resolved and what kind of information is necessary in the negotiations.
8	What are the types of conflicts that you encounter while negotiating with customers on priorities?	
9	What kind of information that can support you in your negotiations and persuasion of customers?	This question covers theme 3 because the feedback as a tool for both end-users and engineers to communicate with each other was to be designed. So there was a need to investigate how engineers would use it too and types of details they would use to negotiate problematic situations with end-users.
10	Do you have any comments, suggestions or advice about our work that you would like to share?	This question covers all themes as it is open for participants to add comments on any of the topics discussed.

5.4.2 Decide on the method of data collection and the audience for the interviews

Ten face-to-face interviews were conducted in the company. The interviews were held in a meeting room to avoid disturbances and have more flexibility and space for discussion. The interviews were audio recorded with consent from the participants, which led to more

flexibility during the interview time because all the time was dedicated to the questions and elaborations with no need write all details. Also, it helped a lot in the transcription process and providing the possibility to be reviewed by other researchers. Each interview lasted average of 30 minutes.

Four different roles were interviewed, which are:

- **Consultants:** this was one of the most important targeted roles in our study because their position responsibilities involve working closely with customers, which include: 1) meeting with customers to determine requirements; 2) clarifying a client's system specifications, understanding their work practices and the nature of their business; 3) communicating with staff at all levels of a customer's organisation; 4) developing agreed solutions and implementing new systems; 5) preparing documentation and presenting progress reports to customers; 6) organising training for users;

  All these responsibilities are core communication tasks with the customers and contain all important inputs for the whole software process and are a main source of problems especially in the maintenance and support phase as it discloses several types of miscommunication and missing information. They also understand different levels of customer's experience, knowledge and problems that could accordingly evolve.

- **Software Engineers:** their work responsibilities include: 1) investigating current applications; 2) communicating with users; 3) producing specifications; 4) ensure that products and enhancements operate satisfactorily; 5) handling support and feedback.

  This role is important in the study mainly because they analyse the customers' feedback and try to resolve their issues based on the input they provide, and thus they are the most affected by the weak feedback structures and acquisition methods. Thus they can provide us with more details of the problems and how they affect the evolution process tasks.

- **Team Leaders:** 1) although the responsibilities are primarily technical, team leaders also serve as an interface between the developers and management, have ownership of development plans; 2) Team leaders also serve as technical advisers to management and provide programming perspective on requirements.

  Most importantly team leaders were interviewed to provide insights about how the evolution process tasks that are affected by the lack of information from end-users and the lack of requirements model employment during the evolution process tasks. Technical team leaders can provide more details about the hidden problems that

customers do not see but are affected by during the maintenance phase, which sometimes influence their acceptance of the software.

- **Project Manager:** 1) They organize, execute, and deploy complex software development projects ranging from short term client specific deliverables to multi-year product roadmap realization; 2) Also, they engage in regular, effective communications with leadership, stakeholders, and internal team members, to ensure they are constantly apprised of all aspects of their project(s); 3) Communicate the benefits of the study/project findings and milestones.
  Program managers were important to provide us with high level problems that occur in planning and prioritization, and customers' business and value understanding. It is important to understand such problems, because if these inputs are properly modelled to represent both the functional and non-functional requirements of the system, and by relating both the structured feedbacks will help them to conclude different levels of evaluation information that can inform the maintenance decisions.

- **Architect:** 1) They subdivide a complex application, during the design phase, into smaller, more manageable pieces; 2) Grasp the functions of each component within the application; 3) Understand the interactions and dependencies among components; 4) Communicate these concepts to developers.
  Architects were important to interview, because they serve as an important link between consultants who gather requirements and developers who implement these requirements. They perform an important task in dividing components and planning packages which is communicated with the customers. Also, they perform technical analysis tasks during the maintenance phase like analysing the impact of changes on the system, and therefore, capturing their opinions about the role of requirements models in facilitating this task and current issues they have is essential to this research. Finally, to capture their current practices and issues regarding how documentation information is captured during the maintenance phase and how they maintain it up-to-date.

A summary of interviewees' roles and experience is listed below in Table 8. Also, it is worth noting that some of the interviewees have multiple roles which is an added value, because they give different perspectives to the problems and situations that happen during the maintenance phase, which enriches our findings as it captures more diverse viewpoints.

Table 8. Interviewees' Roles and Experience

Interviewee	Role(s)	Experience (years)
1	Senior Software Engineer	8 years
2	Architect	10 years
3	Consultant	4 years
4	Project Manager	10 years
5	Consultant / Team Leader	6 years
6	Project Manager	8 years
7	Consultant	3 years
8	Architect	11 years
9	Software Engineer / Internal Project Manager	5 years
10	Team Leader / Project Manager	7 years

5.4.3 Prepare the interview schedule, considering content, wording, format, and structure

For this study an introductory session was performed, where an overview of the research, the results reached so far and its need for this study, its importance and purpose of the study was introduced. This also helped in introducing the interviewer to the intended audience, and also getting to know the participants, their roles, their experience in the company, and also their scheduling preference for the interview session. After this introductory session, issues to consider were:

- Can the question be clearly understood?
- Will interviewees be cooperative in providing the information?
- Is the question related to all interviewees?
- Does the question allow interviewees to offer their opinions/expand on basic answers?
- Will it be uncomplicated to analyse?

After the introductory session presentation, an open discussion was held that helped deduce judgements to the above concerns. Participants shared their opinions about how the topic speaks about their daily problems with customers in the maintenance and support phase. They pointed out that “the questions were organized in a chronological manner starting from the feedback sent by the end-user, going through its effect on all the different phases of handling the software change and evolution (i.e. interpretation, analysis, planning, implementation, and documentation)”. Furthermore, to help organize the content that will be gathered during the interviews and to ease and improve the analysis of the results that will

be collected, the questions were mapped to an initial set of themes that were identified as the main goals from this study as shown in Table 7. This facilitates and emphasizes the identification, examination, and recording patterns (or "themes") within data when conducting the thematic analysis. The themes become the categories for analysis. This supports the familiarization with data, eases the generation of initial codes, searching for detailed themes among codes, reviewing themes, defining and naming themes.

5.4.4 Test the interview with colleagues or potential interviewees and revise as necessary

A pilot interview was conducted with a software engineer in the company in order to test the feasibility of the study (Turner III 2010), providing training to the researcher conducting the interviews, and determine whether the time taken to complete the interview is reasonable or not. Also, the participant was asked for feedback to identify ambiguities and difficult questions. Moreover, it helped in assessing whether each question provides an adequate range of responses or not. The final purpose was to verify that replies can be translated in terms of the required information and themes. This interview was not analysed and included in the study results.

After this pilot study actions were taken to improve internal validity. Two main issues that resulted from the pilot study was that more questions were added to elaborate on important issues regarding problems in their current data collection tool, and how they utilize and reuse the feedback they collect to support them in decision making.

Since semi-structured interviews were conducted, extra questions were adjoined to this main set in some of the interviews especially with the consultants, because they work in partnership with customers (i.e. end-users), advising them how to use the software or overcome problems. This was to elaborate on some specific issues regarding the mapping between the developed structured feedback types and the real situations that engineers encounter, and also to further investigate how the structured feedback types could be utilized to deduce useful information that can inform the engineers during the maintenance a support phase especially when they manage software changes requested by the customers. Examples are:

1) Does JIRA require minimum information to be entered by customers when they report issues? Or is it left open depending on the customer?
2) What are the criteria upon which the consultant qualifies whether a defect is valid or not?
3) Are the problems communicated with the customer in the perception and reproduction phase?
4) How do you keep track of historical data of problems and solutions?

5) How can duplication happen when you have the same problem posted before with a response?
6) Did it happen before that you have suggested a solution to a problem and the user was not able to apply this solution?
7) Do you encounter that users read a feedback thread explaining their same problem and tried the solution that you provided, but did not resolve their issue?

5.4.5 Conduct the interviews

In the introduction, the interviewer:

- Reiterated the purpose of the research by distributing the participant information sheet at the start of the interview and clarified any gaps or misunderstandings about the research topic. The purpose of the study is to gather information about the problems that face the software engineers during the maintenance and support stage. From the point of view of software engineering, software maintenance is the central stage of the software lifespan; for typical successful software, an overwhelming amount of time and resources are spent in this stage and hence it merits particular attention of researchers. The evolution process is initiated by change requests triggered by customers. Our study focuses on identifying the problems caused by the miscommunication between engineers and customers in this process. The interviews are intended to gather information about different problems that the engineers encounter during: gathering information, analysis, prioritization, estimation and so forth. Information collected from the interviews will assist in gaining a deep understanding of how the instances of formalized feedbacks entered by the end-users can be utilized to resolve evolution tasks problems. Thus, direct feedback from interested and concerned individuals is fundamental to achieve this part of the research.
- Reiterated the classification of structured feedback types developed in this research, which is a key aspect in this study, because confirmation is needed from the interviewees that they cover the essential information they need in their communication with end-users. And also, how and when they could be utilized in the change identification and evolution process tasks.
- Recorded interviews with consent from the participants to allow greater interaction between the interviewer and respondent.
- Confirmed with interviewees that their confidentiality/anonymity will be respected. Meaning that their names will not be linked with the research materials, and will not be identified or identifiable in the report(s) that result from the research.
- Confirmed with interviewees that data gathered from the results of the study may be presented at a conference or published, provided that they cannot be identified.

- Kept the interviews as planned in place and time according to the schedule that was pre-prepared and agreed upon in the introductory session.

5.4.6 Transcribe interviews

A transcript that contains text that describes the content of audio or video files was prepared after the interviews were conducted. These transcript copies were used to:

- Make a text copy of all interview discussions
- Divide the audios into sections and make notes about each section
- Record notes about specific ideas or questions
- Make notes about the interview audio in general

All audio files were imported on NVivo 11 for transcription. A transcript consists of the time span selected from the audio file and associated with the translated text for this time span. After the transcription of interviews the analyst can:

- Add, delete or edit the text in transcript rows
- Format the text in the Content column
- Code, annotate and link text in the Content column to memos
- Filter the transcript to focus on pertinent content
- Select an entry and play the associated section of media

5.4.7 Analyse the transcripts

The interview transcripts were analysed by using thematic analysis method. Thematic analysis is used in qualitative research and focuses on investigating themes within data. This method highlights organization and rich description of the data set. Thematic analysis tries to find implicit and explicit ideas within the data. Coding is the primary process for developing themes within the raw data by recognizing important phrases in the data. The justification of these codes can include comparing theme frequencies, recognizing theme co-occurrence, and graphically displaying relationships between different themes.

The analysis resulted with two thematic maps representing the classification of missing information that affect the maintenance /evolution tasks shown in Figure 12, and the classification of problems triggered by the missing information that arise during the maintenance and support phase shown in Figure 13.

5.4.8 Write up, present and use the findings

The study results will be explained in detail in section 5.5, and how the findings will be used to achieve the objectives of this research will be explained in section 5.7 of this chapter.

5.5 Study Results

In this section the study results are described in detail. The results are categorized into the two main categories: 1) the types of missing information (Section 5.5.1) from the users' that can lead to several types of problems during the maintenance phase, which is represented in the thematic map in Figure 12; 2) the change identification and evolution process tasks' problems (Section 5.5.2) that engineers meet while handling change requests represented in the thematic map in Figure 13.

5.5.1 Types of Missing Information

This section explains the types of missing information that were extracted from the interviews and classified as mandatory and useful information necessary for proper execution of the evolution process tasks. Improper representation and utilization of these information categories hinders the engineers' capability in resolving continuous rising issues during the evolution process and decision making.

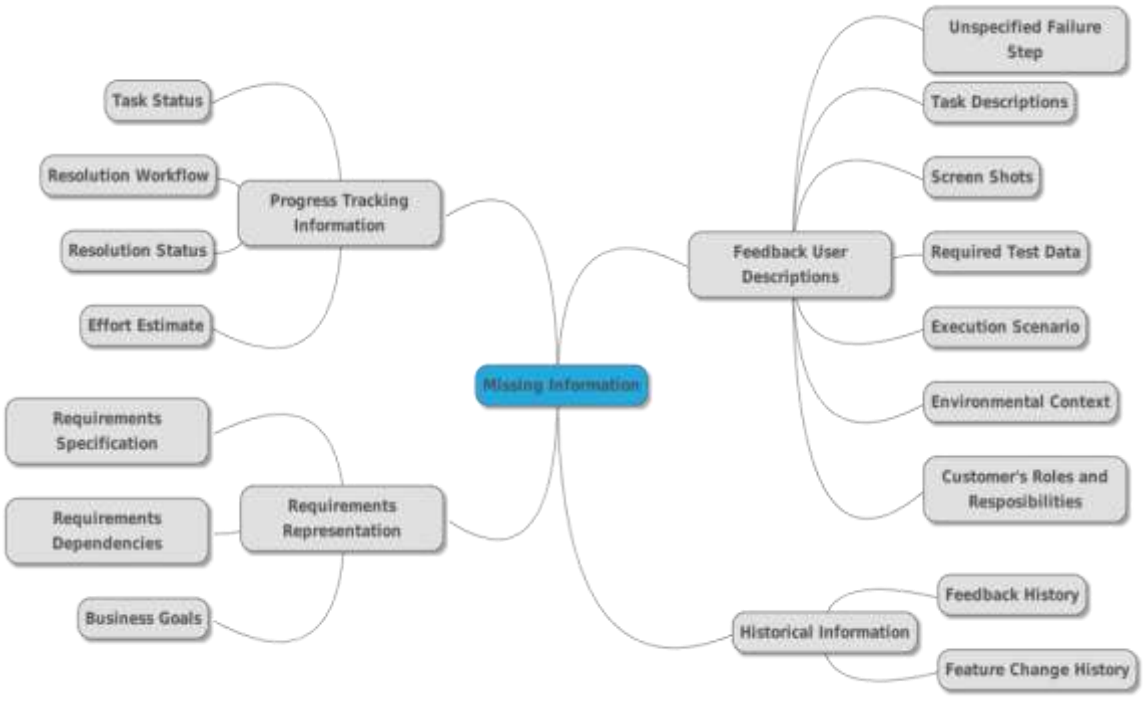

Figure 12. Interviews' Study Final Thematic Map for the Types of Missing Information

5.5.1.1 Requirements Representation

Successful projects engage users early and then discover and attain closure on their requirements by using analysis models -- representations of user requirements. Despite of its importance, there is lack of formalized method and frameworks that aid software engineers in integrating these models with evolution tasks. This thematic area incited in the

interviews as a main concern during the evolution process tasks, because consultants and customers depend heavily on it in validating issues, also in narrowing down problem scopes, and in determining the impact of a change when deciding on new enhancements or features. Therefore, the need for formal, organized usage of updated requirements' representations is classified as mandatory in this research.

5.5.1.1.1 Requirements Specification

Requirements Specification captures the need for proper specification and the need for keeping specifications up-to-date along the maintenance and support phase for accurate handling of software changes. Specifications are important to determine the features' definition, scope, and behaviour. This information is an important input for the identification task in which the engineer determines whether the feedback is a help request, a change request, a new feature request, or a bug fix.

For example, sometimes due to the customer's inexperience with the system, he may request a feature that already exists. So engineers need *"customers to link the reported defect to one of the requirements that are already written in the requirements specification, because this helps us analyse the use and revise its scenario to evaluate whether this is a valid defect or not."* Engineers also explained how specifications could be used to determine new change requests *"If the customer relates the problem to a feature in the requirements document then it's a defect. If not then it is a change request. Also, change requests can relate to a feature in a requirements document but with a different behavior."* Another example: *"After we revise the specs written in the file concept, if we do not find the requested functionality then we can convert it to change request and make an offer to the customer including costs."*

5.5.1.1.2 Requirements Dependencies

Requirements Dependencies emphasises the need for a proper capture, storage and representation of dependencies between requirements. Requirements and/or feature dependencies are very important in the analysis task especially that they are the main input for studying the impact of a change, because "*if we can locate the requirement/feature that requires the change then it can be examined whether the related requirement(s)/feature(s) will be also affected or not"*. It also helps in producing better estimates by the software engineers during the planning tasks.

In the study interviewees confirmed that this task depends on several team members' experiences and is done through manual inspection *"The consultant who is aware of the impact of changes, because he knows the requirements and the relationships between them, and the developer, because he knows where a function could be usable from several places in the code."* That is sometimes error-prone and inaccurate especially if the team

members are new to the project and unexperienced with the code as stated by a team leader participant: *"Usually this problem occurs with junior software engineers, because they try to resolve exactly what the customer needs, without looking in different areas to provide a complete clean solution."*

5.5.1.1.3 Business Goals

Business goals representation can help engineers relate requirements and features in a meaningful way without losing the big picture overview. It can help management team members gain a clear picture of customers' goals and priorities. Lack of business goals representation may inhibit ideas about future possibilities for change requests, especially in systems operating in highly variable contexts. A project manager participant explained that *"Sometimes changes are very important to users and they do not cost us much, and the opposite way around, sometimes less important features may cost much effort."*

Currently interviewees have pointed that communicating new change requests is dependent on the project managers' experience and also to the consultants' knowledge with the customer and history of similar cases related to him subject to manual research. *"Technical estimates are given to the consultant to revise the history of previous change requests to that customer, and the requirements document to be able to estimate its value to the customer."*

5.5.1.2 Feedback Descriptions

Feedback descriptions constitute a thematic area in which the essential components of feedback is captured and explained. The need for software maintenance and evolution is triggered by end-users' changing needs and/or problems that are communicated through feedback while the software is in use. Therefore, the better descriptions they provide the better the problem or change request is communicated and interpreted leading to successful implementations and closure.

In this study it was also explained that feedback is not only used by end-users to communicate their issues, but also by engineers in communicating with end-users and/or in resolving their issues. Thus, feedback is used as a communication method. This highlights the importance of providing enough details to ensure the feedback conveys useful and meaningful information.

5.5.1.2.1 Execution Steps

As concluded from the forums analysis study that explaining through steps is a powerful tool when describing both problems and solutions. From an end-user side, when a Problem feedback type is provided, a Trial level of detail could be used to describe the steps that the

end-user has gone through in performing a certain task. A trial level of detail should be explained using steps to provide engineers with guidelines that enable them to interpret and reproduce the end-user's problem: *"Examples of problems that can occur are that users do not write a complete scenario (steps) to the problem so I do not understand how the problem occurred to him."* Also, *"Detailed processes can become confusing if conveyed in paragraph form. Writing in an ordered list instead, makes following directions much more clearly to the engineer."* The customer can also quickly identify where they ran into trouble in a process. *"For example, "Step three gave me error 33."*

From an engineer side, when a Mitigation feedback type is provided, then it should contain a Usage scenario level of detail that explains the solution in steps to enable the customer to try the solution easily and locate any problems accurately.

5.5.1.2.2 Unspecified Failure Step

Explaining the Problem feedback type in a form of Trial steps was pointed by different team members as an important factor of understanding a reproducing the customer's problem. However, in some situations this is not enough it also has to be accompanied by exactly specifying the step in which the problem occurs. An example from what consultants has explained: *"We start getting into more details with the customers…So he starts explaining a scenario including details about the task, and when exactly does the problem occur."*

Lack of this type of information can lead to the inability to design a proper configuration and solution. For example: *"Say a feature is not working, and it can be reached through several ways in the application, which way or scenario was the customer using and when the error has occurred is important information. This lack of information is also misleading sometimes"*

5.5.1.2.3 Task Descriptions

It captures what the user was doing when the problem occurred. This is specifically important when the user is describing a problem, because it gives the consultants and software engineers an idea about the context in which the problem occurred, or describing the frequent jobs that the user is involved in in his daily work. For example, *"A customer gives feedback saying that he was doing an import and import failed."* This problem lacks task description that describes the job the end-user he was trying to accomplish. *"So we start asking him more questions like: which type of import was he using, why was he using it…"* This information helps in *"better understanding the problem and in narrowing down the scope and deciding where and when the feature simulates a problematic situation"*

5.5.1.2.4 Screen Shots

The text method is the most commonly used description method in all feedback types. However, sometimes there are problems that users find hard to explain using textual information alone, and they use other supporting methods such as screen shots. *"Sometimes customers provide very brief descriptions that become more understandable when providing supplementary materials such as a screenshot".* Screen may help in illustrating *"the feature he has been using, the task he has been performing, and/or the error message that occurred to him".* However, using screen shots alone are not enough to convey all information about a problem without explanation. *"The error alone is not enough, because he has to provide other descriptions and other information such as the data he was working on."*

5.5.1.2.5 Required Test Data

Test data is a new level of detail that was discovered from this study. It is an important input that customers can provide especially when the problem is related to data-oriented features, such as exports and imports. *"If a customer reports that a problem occurred while importing data from a certain file, then he might be asked to provide the file for us to try on."*

Lack of this information may lead to inability to identify or reproduce problems, because *"If the customer did not provide such file the consultant may try to create some mock data and try it on the system for testing. Of course the mock data may not reproduce the same problem as the real data."*

Also, sometimes engineers *"cannot make accurate expectations of the numeric results that will occur.* And sometimes when they resolve an issue and correct values occur, *"customers start reporting that the next step doesn't provide correct calculations."* Thus, providing an actual set of data used will help save time and effort that would be spent by the engineers to produce mock data to make trials and try to reproduce the same sequence of errors that occur at the customer's site. *"Inconsistency between the data may make the solution inoperable".*

5.5.1.2.6 Environmental Context

Environmental context category is a level of detail that captures context information related to a software or hardware specs, versions, configurations…etc. Participants elaborated that *"end-users can provide these kinds of information in a problem statement to specify the software version they are using, which may differ in the problematic feature from older or newer versions."* Therefore, this adds specificity and usefulness to add such information.

Also, participants pointed out that when they provide solution to the end-users too this is very important information that has to be clearly stated, because *"It might happen that the user cannot try the solution we gave. This can be because we were not considering an aspect in their environment."*

5.5.1.2.7 Customers' Roles and Responsibilities

This conveys context information related to a user's role at work, information about co-workers…etc. This was already identified in the forums analysis study as a context category named Social context. Participants in this study elaborated on its use and importance such as *"Which user role was used (logged in) during the import operation? As the problem might be in the user rights that doesn't allow them to perform certain tasks on the software"* Therefore, this information is important to determine which action should be taken in the solution design.

Also, another use of information on roles and responsibilities is: *"Sometimes customers specify the number end-user roles getting a certain error when using a feature in the application to emphasize the importance of resolving it, and giving it a high priority."*

5.5.1.3 Historical Information

Historical information is a thematic area that refers to any information encompassing resources that may be employed by a project team and/or a project team leader for the purposes of gathering as much information as possible about projects, activities, or events that had taken place in previous time periods for the same project with the intentions of delivering sufficient insight and background to the team when making decisions that will eventually affect the resolving of current issues on hand.

5.5.1.3.1 Feedback History

Using stored structured feedback threads containing the feedback types that were utilized in the communication between end-users and engineers to resolve an issue, enhancement or new feature that took place along the project *"could help provide future insights in other issues during the same project or any future similar projects"*.

The lack of feedback storage and feedback linking hinders the engineer's search capabilities in the historical information of the project. Thus there can be no historical information reuse, and problems such as duplicates cannot be systematically detected, as consultants rely on their experience and knowledge *"As a consultant if I am involved in-depth within the project, then I can easily identify duplicates."*

5.5.1.3.2 Feature Change History

Feature change history expresses the necessity of keeping requirements information up-to-date and keeping track of feature changes, how, and why they occurred. *"Keeping this information up-to-date is mandatory when applied in the context of change impact analysis to produce accurate results when analyzing future changes."*

Participants also pointed out that *"keeping track of the changes that occurred on a certain feature can be useful when analyzed in future improvements suggested by the software development team to determine its value to the customer"*, which will provide better support for decision making. However, this is currently performed manually with no information support for the consultants: *"Technical estimates are given to the consultant to examine the history of previous change requests for that customer, and the requirements documentation to be able to estimate its value to the customer."*

5.5.1.4 Progress Tracking Information

Tracking information is important in any software project because it helps team members stay focused. Many project team members often get overwhelmed when faced with big tasks. The perfect corrective action is to track the project tasks. Results from each task or step should also be tracked. After that, it can be just a matter of concentrating on the tasks that yield better results.

This thematic area does not capture the types of information that could be found in feedback, but it contains types of information that can be deduced from feedback responses (i.e. threads), or are stored explicitly by engineers. Lack of this category of information can affect some of the software evolution tasks such as planning and adversely influence the communication between the end-users and engineers.

5.5.1.4.1 Resolution Workflow

One of the important tracking information is the resolution workflow, which captures the steps that was followed by the software engineers in order to resolve an issue reported by the customer. Lack of capturing and storing this information can cause deviation from settled targets, and communication problems with customers that can often lead to customer dissatisfaction.

Participants explained *"We store all problems and their workflow on JIRA: whether they are qualified problems or not, their status, description, the action that was taken for this problem, resolved state, and customers' acceptance whether it is resolved or not."* In large projects this helps in *"providing accurate plans for updating customers with correct and*

exact decisions/ results that were reached". And can be *"further used in project documentation"*.

5.5.1.4.2 Resolution Status

Resolution is an evaluation case that indicates whether a problem posted in a feedback thread was resolved or not. *"Unresolved problems mean that the customer did not verify the suggested solution and therefore, the issue will still be open waiting for a workaround or solution. If the customer is satisfied then the issue is closed and proper documentation is provided. Also, resolved issues can be re-opened for further enhancements".*

This case is captured, because interviewees have explained how this is done on the JIRA tool that they use. However, if a formalized acquisition and communication method will be developed for both customers and engineers, *"this information could be deduced automatically by monitoring feedback Topics and feedback Response types"*. It can also be used as *"success/fail measurement of project or releases"*.

5.5.1.4.3 Task Status

Effective task management requires managing all aspects of a task, including its status. One of the primary concerns of the project manager during the maintenance phase is keeping the issues handled on schedule *"Because customers care much about the due dates in this phase, as they need them to meet certain needs or tasks. In order to do this, managers must identify those tasks that aren't getting completed on time"*. Also, it can be used as base data to infer useful information about *"project success/fail rate and measure customer satisfaction"*.

5.5.1.4.4 Effort Estimate

Effort estimation is the method of identifying the most realistic utilization of effort required to resolve a customer issue. The estimates of the effort might be used as input to project plans, determining the budget and other important procedure required for the successful completion of tasks and releases. Good effort estimation is the one that can be explainable in each and every step.

For example, *"if the customer requests a new feature to be delivered in one week, but due to lack of information in the problem statement we have to investigate more. This sometimes leads to major changes in the feature and transforms it to another different specification with a totally different estimate due to the addition of tasks".* This is an example of a problem case that occurs that makes it a necessity to communicate estimates with customers.

5.5.2 Maintenance Phase Problems

The thematic map in Figure 13 represents the problems related to the evolution process tasks starting from engineers receiving change requests and how they are handled until change acceptance and closure. Also, miscommunication between end-users and engineers was identified as a one key problematic area during the maintenance phase as it captures all the interaction problems and their causes. The problems captured in this thematic area are all consequences of the different types of missing information explained in the previous section 5.5.1 as concluded from the interviews. That is why it is important to summarise these problems, because this research focuses on the capturing and formalization of feedback to utilize it in communication and evolution tasks execution.

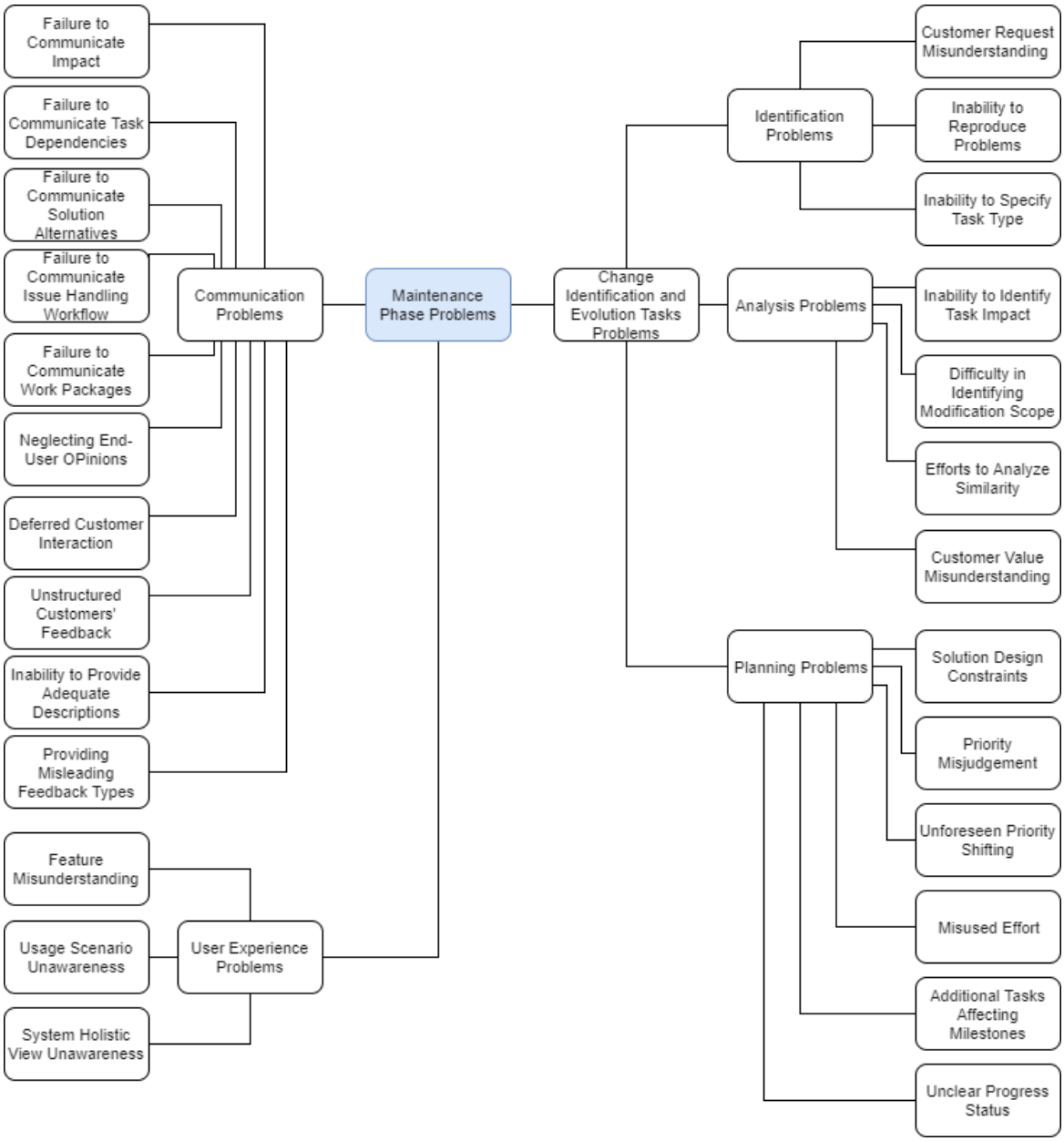

Figure 13. Interviews' Study Final Thematic Map for Maintenance Problems

5.5.2.1 Change Identification and Evolution Tasks Problems

The evolution process is normally triggered by a change request sent by the customer followed by a group of tasks that the engineers perform to manage a change till reaching a successful closure with the customers. Participants stressed on three main consecutive tasks for managing a change after it is received, which are identification, analysis and planning. These tasks are dependent on the utilization of end-user input (i.e. feedback), requirements documentation and modelling. Normally, their successful completion would lead to a successful change implementation and closure.

5.5.2.1.1 Identification Problems

The First thematic area in the evolution problems is the change identification phase. This theme encompasses all categories of problems that relate to the incapability of engineers to understand the end-user feedback, specify the issue type, or reproduce the problem to confirm its validity and proceed in the process to reach a suitable solution. This mainly relates to the lack of sufficient information or model representations that can aid engineers in their tasks.

5.5.2.1.1.1 Customer Request Misunderstanding

Communicating clearly with the customers can be challenging, especially if your interaction reaches into the technical details of your product. This problem is related to a category of missing information named Feedback Description explained in section 5.5.1.2 which is the core reason for customer request misunderstanding.

The core idea is that there are no guidelines for end-users to follow while providing the feedback. Therefore, it is left to the customer's ability to express, which most of the time as reported by the interviewees is very weak, due to the lack of technical background, and sometimes inexperience with the system. *"Sometimes a customer's request can be a bit mystifying. Consultants might make assumptions too quickly about what a customer means and end up trying to solve the wrong problem."*

There are types of details that are preferable to be used in feedback descriptions. When used correctly with enough information, to provide a useful and meaningful input for engineers can lead to better understanding of the problem, examples include:

"Screenshots are a great thing to request from customers. A screenshot can quickly clear up a misunderstanding that might otherwise take two or three emails to clarify in text."

"Detailed processes can become confusing if conveyed in paragraph form." "Try using an ordered list instead. This makes following directions much more clear." The engineer can

also quickly identify where end-users ran into trouble in a process. For example, *"Step three gave me error 33."*

5.5.2.1.1.2 Inability to Specify Task Type

There are several types of tasks that customers report to engineers either defects, enhancements, or new feature. First task the consultants do in the evolution process is trying to identify the task type or trying to verify whether the task type that the customer specified in the feedback is correct or not.

The inability to specify task type may be due to several reasons either the customer provided a misleading type due to lack of experience or conflict in understanding the existing software features *"This usually occurs in the beginning of their usage, as users are still not experienced on doing tasks using the software. So they cannot judge whether it is the correct behaviour or it is a mistake in the application."*

Furthermore, insufficient information can hinder the engineers' capability to specify the correct task type. *"We continuously ask customers to link the defect to one of the requirements (i.e. use cases) that are already written in the requirements specification. This helps us analyse the use and revise its scenario to evaluate whether this is a valid defect or not."*

5.5.2.1.1.3 Inability to Reproduce Problems

Reproducing problems means the defect/bug reappears at the development environment then engineers can resolve and fix the Bug. However, *"if the bug is not reproducible at the development environment then more investigations are sent to customers to gather sufficient and/ or correct information".* This is the first step in a problem identification/validation.

"Sometimes users do not give enough description to the problem, and they only give a screen shot. Also, they do not describe the scenario and context in which the problem occurred." Thus, the user's problem is not reproducible locally in our environment, and is only reproducible on his system. *"Mapping between our configuration and his becomes hard".* This Lack of information in customers' feedback is a main cause for unsuccessful reproducing of problems, which can be very time consuming for engineers, and can affect the subsequent tasks in the evolution process causing higher costs if identified later.

5.5.2.1.2 Analysis Problems

The evolution process includes fundamental activities of change analysis, release planning, system implementation and releasing a system to customers. The cost and impact of these changes are assessed to see how much of the system is affected by the change and how

much it might cost to implement the change. If the proposed changes are accepted a new release of the system is planned.

5.5.2.1.2.1 Difficulty in Identifying Modification Scope

This is an issue that engineers encounter during the analysis task of any new issue. That is how the change is going to propagate and affect other features and/or requirements in the system. This is a key task in the evolution process and that is dependent on the category of missing information identified in section 5.5.1.1 named "Requirements Representation". Therefore, it must be kept up to date with enough information and descriptions for each requirement and its related features for the engineer to be able to use it as an accurate reference in analysis tasks. Engineers suggested that this could be linked to end-user feedback by *"Asking The user to reference the components of the software in his feedback to determine from the user's point of view in which component the bug exists. This also helps us determine the scope of the bug and the type of modifications needed"*

This also helps engineer to analyse whether the issue is a bug or a change in the functional definition of the feature, its behaviour. Participants provided insights on the important of linking the feedback to specification to be able to trace the issue and analyse it *"We have included a field for the chapter in the requirements specification that includes the issue. This is because many times there are conflicts between us and the users on whether what they are requesting was already in the specs (therefore it is a bug) or not."*

5.5.2.1.2.2 Inability to Identify Task Impact

Task impact is identifying how the customer request will affect other parts in the system. Participants said that two main roles are responsible for examining the impact of a change *"the consultant who is aware of the impact of changes because he knows the requirements and the relationships between them";* and *"the developer who is aware through the implementation of the modules where a function could be usable from several places in the code."*

However, this task is done manually and depends on the consultants' and developers' memory and experience *"During the estimation time, the one who was doing the estimate should have an overall view of the system, and he should be given enough time to analyze and determine an accurate impact."* Also, this task is dependent on the Ability to identify the modification scope correctly, because if the engineer locates the requirement that needs modification, trough requirements and feature dependencies the impact could be identified more accurately producing better estimates.

5.5.2.1.2.3 Customer Value Misunderstanding

The most prevalent misunderstanding of the customer value concept is that *"value means low price, or minimizing its scope to obeying what customers want, being nice to them, or delivering outputs as fast as possible."* However, managers pointed out that it is as much about being proactive in coming up with new products/services/experiences that they believe will create value for customers.

"The role of the good consultant is to find this "sweet spot" of providing to the customers what they want without great deal of IT effort. So if a conclusion was reached with what the customer needs is very high in cost, which is impossible for him to accept, *"then we start suggesting alternatives with some functional limitations, restrictions or workarounds. And while negotiating with the customer we start understanding its importance and value from him."*

However, this depends on the consultants' and managers' experiences and knowledge with the customer's needs, goals, and domain. *"So it is given to the consultant to revise the history of previous change requests for that customer, and the requirements document to be able to estimate its value to the customer."* Lack of historical information about feedback history in dealing with customers' change requests and the representation of the customer's goals related to the requirements and feature affects the big picture summary, which is important in understanding customer value and decision making.

5.5.2.1.2.4 Efforts to Analyse Similarity

This is a very important issue that specifically concerns consultants who are responsible for validating customers' reported feedback. After a period of time in a project the number of feedback from customers and responses may increase a lot, making the analysis or search task much more complex and time consuming especially if it is done manually or with very limited capabilities. Consultants mentioned that they *"link similar issues together manually. If we find a similar resolved problem we refer to it by replying see issue number X, and the new issue is closed as a duplicate."*

The lack of proper linking between feedbacks and depending on human capabilities in handling similarity analysis *"The feedback is analysed and if the consultant realized that this issue is repeated we report that it is a duplicate and he refers to the older issue number"* This might lead to error-prone results, such as inability to identify duplicate problems, which leads to duplicated effort. *"The problem is if he does not remember we will go through the whole process again until it is resolved with this user."*

5.5.2.1.3 Planning Problems

Before the actual task(s) implementation begins, team members must make sure that the work is properly understood and agreed to by the team members and the customers. Planning is one of the most important project management and time management approaches. Planning is preparing a sequence of action steps to accomplish some specific goal. Putting a plan requires team members to put an initial solution design and break it into smaller tasks with priorities, and make estimates of time and cost. If team members follow it effectively, they can reduce much the necessary time and effort of achieving the goal. When following a plan, engineers can see how much they have progressed towards the project/ task goal and how far they are from their submission.

5.5.2.1.3.1 Priority Misjudgement

The lack of engagement with customers, learning from their work usage habits, and previous problems or needs may cause disagreements and misjudgements about priority *"Usually customers enter the issues in the order they would like us to address them. However, if we are not convinced with their way in prioritization, then we ask them for more information."*

Sometimes they request issue that seem simple or unimportant with high priority therefore more background information is important to justify, why they are requesting the issues in that order. *"For example, sometimes they have an audit from an external company X, and they will lose a license if there are certain defects in the outputs from the software."* Also, customers can set false priorities on the problems or changes they request *"due to lack of experience and knowledge with the system and its related functionalities"*

A team leader interviewee also mentioned that *"Priority also depends on the Importance or the urgency of the problems."* For example: *"A customer is in a central organization with many end-users that work on an application that we have developed. Once they were generating a certain report, and errors occurred, so they reported to me the errors with details".* There was lack of information about the Requirements specification and its dependencies *"from which we can estimate the roles that use these requirements and the estimate of the number of affected users and the estimate of its frequency of occurrence from the technical design"*.

And thus all this lack of information may cause the priority misjudgements and inability to identify an accurate impact and also failure to communicate that impact with the customer, and waiting for the problem to evolve at the customer's site, as *"The more users that report that they have the same problem, the more the issue took a higher weight, because it affects the quality of the produced reports."*

5.5.2.1.3.2 Unforeseen Priority Shifting

One of the most problematic issues about priority lists are they seem to grow and mutate constantly. At some point during a project a customer is likely to make a demand that shifts priorities; or, another team member's work is delayed and that affects the flow of the plan.

Unforeseen priority shifting can be caused by *"lack of information in new change requests that comes in from the customers about its urgency or importance, which leaves it to the engineer to prioritize it according to the settled plans and available time and resources"*.

Also, *"lack of information representation about customer's goals related with requirements specifications that can help engineers identify the value of a change can lead to false priority judgements and thus leads to future shifting in plans and less customer satisfaction".*

Other Factors that obstruct our plans include: *"we do not have much available time, bugs that occurred are more than expected, or requirements change...etc. So in these cases we need to re-prioritize our plans. If not this extends the deadlines a lot, which puts much emphasis on the prioritization of issues."*

5.5.2.1.3.3 Additional Tasks Affecting Milestones

An additional task affecting the milestones is a result of other planning problems, whether prioritization misjudgements, design and/or deployment constraints, or unpredicted priority shifts. All these problems will definitely lead in the end the state of updating the project plan and affecting milestones. For example: *"Sometimes the customer reports an urgent issue so we spend time and effort to give him a temporary workaround that might affect the current plan. Then we explain to him that in order to provide a better solution other feature(s) need update first so we have to delay it to another release if possible"*

To avoid the ad-hoc changes in the plan, the interviews pointed out this this is mainly due to miscommunication of rising problems with the customers. Pinpointing the consequences of a problem is important to help us avoid these situations and resolve its causes. Sometimes tasks in the evolution process take longer than expected *"even if we provided the fix in a timely manner. But in the end there are some steps that we cannot eliminate."*

Engineers do not make definitive decisions, but instead they support the customer with information (resources, deadlines...) and concerns. For example, *"in order to update the production system we have to shut it down and we cannot do this while the end-users are working, so we have to choose an appropriate down time, after having this permission even if it is at night while they are not working on the system."*

5.5.2.1.3.4 Misused Effort

All of the above problems that were discussed end up as tasks that engineers have to perform, the more problems that occur in these evolution tasks the more the wasted effort. An example of misused effort is the effort spent on investigations due to lack of adequate description from customers. Examples include: *"Customers do not give lots of information in their feedbacks and thus the communication time is longer and consequently the estimation task."*

Another example of misused effort is the effort spent in implementing workarounds that will be changed due to unforeseen priorities. *"There are clean solution and workarounds. Some situations may lead us to do workarounds especially when the problem is a blocker for example in a production system and we have to solve it immediately."* By the code below it is meant that there is effort wasted in designing crooked workarounds for example in blocker situations and then spending more effort in designing and implementing a clean solution in the future. *"Usually after this kind of workaround we must find and implement a clean solution. Sometimes problems are solved through workarounds that require a change in the workflow of the task."*

The following case was coded under misused effort, because the interviewee said that instead of asking customers several times for the same information and trying to narrow down the problem with him, they start in making several trials which is a waste of effort. "For example, *"a user reports that an error occurred when he entered certain data. He has to provide the data that he entered, because some problems are data-specific. Sometimes he provides missing data, old data, and/or wrong input data."*

5.5.2.1.3.5 Unclear Progress Status

One of the key factors of why projects fail is related to visibility. All team members need access to the right level of information at the right time to be able to successfully manage their tasks and communicate with customers when needed. There is a lack of systematic tools and process for handling these issues *"We substitute the lack of formality with consultant's experience with the functionality, and the technical knowledge of the architects and their experience with the code and its updates."*

There is extra effort wasted in communicating information between team members. *"Internally, we make weekly meetings to discuss the issues and their progress, their implementation and how it was done and the updates. These meetings are the best time for the tester or the consultant to know which parts exactly are affected with the updates and needs testing."* Also, sometime by the time the team members call for a meeting, *"some members are working on past issues that have changed scope or evolved due to continued update requests."* Therefore, devising a communication means that enables every customer

update to be documented, and related to the feedback thread for a main issue is important to keep information linked and accessible for team members working on the issue and/or project.

5.5.2.1.3.6 Solution Design Constraints

Covering design inputs in solution design is very important; design inputs are Requirements, Constraints, Assumptions and Risks. It's important to classify design Inputs, and that is why these codes were captured from interviewees answers that pinpoint the important information that has to be communicated from customers or other team members to produce better solution alternatives to customers. Examples of information the end-users should provide in their feedback are: Priority and Severeness of an issue. "*For example, if the customer reports a blocker problem and he needs to resolve it maximum tomorrow. Therefore, we will do a workaround until a clean solution can be made."*

Also, the budget is an important factor *"whether the customer has no budget or not willing to pay for a clean solution, or we have no resources on the project to make a clean solution. Out of honesty we inform the customer that there is a clean solution, but due to budget issues we will make him the workaround."*

Moreover, Quality is an important factor; *"if we have to consider quality then we have to go for a clean solution. If we will not consider the quality because it is an unimportant feature that is used annually then we can make a workaround."*

Furthermore, knowledge about the roles that will use a certain feature or module is important *"to design the entrance points from different parts in the software. We might also want to know the number of users to handle performance if there will be a large number of concurrent using the feature at the same time."*

5.5.2.2 Communication Problems

By communication means the reporting or exchanging of information between the engineers (the software company) and the customers (end-users). This category encompasses miscommunication categories from both sides either they communicate mistakenly, unclearly, or inadequately. Also, the communications problem is further complicated by the number and diversity of system change requestors.

5.5.2.2.1 Unstructured Feedback

Unstructured feedback refers to the problem that there are no pre-defined rules for the mandatory components of a feedback that can act as a guideline for end-users. Unstructured feedback is a core problem as it causes difficulty for engineers in interpreting them and reaching useful or meaningful information that can inform the

evolution tasks. Also, to resolve this issue a considerable amount of time and effort is spent in investigation. *"Each defect has a summary as short text, and a description as free text to write what he wants. The JIRA tool allows for the above mentioned details to be entered however it does not assure that the customer enters a minimum amount of valid information."*

"Normally, in all our projects we do not set restrictions on the minimum amount or types of information that the customer should enter." Although, JIRA allows setting restrictions like not allowing customers to enter issues without providing descriptions *"but we put these restrictions to customers in form of a gentle agreement to provide this information."*

Also, some of the interviewees provided examples indicating that there is no clear structure for end-users to follow when providing their feedback. If engineers still do not understand the problem well, *"then we respond by feedback required from the customer. We provide some questions and or request files"* This means that the issues are totally handled case by case through investigations without any predefined rules to govern the structure *"and customers keep responding until things are clearer then things get back in progress and we start working on resolving it."*

5.5.2.2.2 Providing Misleading Feedback Types

Misleading Feedback type means that customers tend to give false ideas or impressions about the type of problem they are describing either deliberately or unintentionally. *"Usually first thing that the users do is that they create an issue on that tool. This issue has a type: enhancement, defect, feature, or change request. Typically most customers report any problem they have as a defect, because defects are resolved for free."*

This can be either due to lack of training on what the problems types they can report are and how they can report each type, or lack of experience with the software.

Also, lack of sufficient descriptions in feedback can mislead engineers in different types of tasks. Lack of updated requirements information can be misleading especially in identification tasks *"Lack of information especially use cases is a root cause of misleading engineers."* Say a feature is not working, and it can be reached through several ways in the application, *"which way or scenario was the customer using when the error occurred is important information.* This lack of information is also misleading sometimes, *"especially if we cannot reproduce the issue on our system."*

Software engineer develop effort estimates that are likely to be highly inaccurate and systematically overoptimistic. *"Engineers suggest that the causes of the problem, to some extent, were due to the influence of irrelevant and misleading information, for example, information regarding the client's budget, present in the estimation material."* Also, unclear

progress status information of some tasks can be *"misleading in scheduling tasks and resource allocation."*

5.5.2.2.3 Inability to provide Adequate Descriptions

Inability to provide adequate descriptions refers to the customers' failure to provide sufficient information enough to describe the problem they want to communicate with the engineers. *"Sometimes the problem is in the data the customer enters, so providing the data could be important in identifying the problem."*

The lack of this information makes engineers *"unable to reproduce the request that the customer sent."* On other times, *"the user's problem is not reproducible locally in our environment, and is only reproducible on his system. Mapping between our configuration and his becomes hard."* So engineers start asking customers more questions like: *"which type of import was he using; provide the import file, which user role was used (logged in) during the import operation as it might be a problem in the user rights."*

This category contains the codes illustrating the problem existence, and the effects it causes by evolving other problems, also it contains specific types of missing information that were captured in more details in section 5.5.1.2.

5.5.2.2.4 Deferred Customers' Interaction

Delayed Customers' Responses represents a behavioural task in which customers do not communicate the adequate description in a timely manner this can lead to problems in the tasks depending on that input such as the solution design, estimation and planning.

Any change made in further cases will cost more in terms of time and effort. *"It happens that while we are in the middle of the estimation task and after we reached an agreement on a solution with the user that he starts making changes and adding problems. This may lead to deviations from the expected solution."* This leads to more effort exerted by engineers in validating the new added information, analysing its impact, assigning priority, and putting it into plan.

Other cases include when customers communicate their problems shortly before their deadline which interferes with the prioritization engineers have in plan. *"Mainly they are influenced by their own deadlines. If they have a deadline and a feature produces an error making an obstacle for him to meet the deadline, then they put high priority for that issue."*

5.5.2.2.5 Failure to Communicate Work Packages

A work package is a group of related tasks within a project. A work package is the lowest component in a work breakdown structure (WBS), sometimes called the terminal element of

a WBS. You create a work package when you decompose a deliverable into components while creating a work breakdown structure. *"A work package is a way to both understand cost and duration and easily manage those aspects"*. A work package should be unique to the WBS. Without having information about the release work packages, *"customers will not be fully aware about the scope of the release. Consequently, problems would occur during release planning e.g. conflicts in priorities, and task durations."* Also, The customer plans for acceptance tests that will be conducted when the release is delivered. So, *"without complete information about work packages they will struggle in designing their test plans, which will affect their acceptance of the release"*.

Completion of work packages can be dependencies for other work packages. *"We can implement parts from several modules but the customer will not be able to benefit from such release. So it's not by the number of modules we address in the release, but it should also be a working part as a whole."* It is important to communicate work packages with customers, because they understand the software in terms of how it maps to their actual work process that they perform in their daily work. So they need to understand what the scenarios that will be delivered in the coming release. Work packages are also planned according to technical task dependencies which are the engineers' technical perspective.

5.5.2.2.6 Failure to Communicate Issue Handling Workflow

Failure in communicating issue handling workflow refers to the lack of information about how the engineer should respond to different type of requests, and what is the necessary information needed to resolve each case. For example *"If the customer did not provide the import file the consultant may try to create some mock data and try it on the system for testing."* The mock data may not reproduce the same problem as the real data, which will waste effort in creating mock-data and will lead to failure to reproduce the problem due to inconsistency. *"If the analyst had made the customer aware that he will need the real data; this would have helped more in reproducing the bug successfully and saving time and effort."*

5.5.2.2.7 Failure to Communicate Solution Alternatives

This problem represents the engineer's inability to find appropriate solution alternatives to customer requests. Engineers may try to find alternatives due to the impact the change will pose on parts of the system, *"Say for example the customer wants to be able to localize a page. He can do so by letting me edit in a properties file. This is a simple solution, but each time he wants to do this task he has to communicate this with me for the editing and update. Or there is a clean solution; in which we can implement a configurable page to do the task."* Miscommunication of possible alternatives may lead customers to retreat from their decisions regarding their need for an enhancement or change.

Inability to identify possible solution alternatives may happen due to the lack of proper requirements documentation for supporting engineers in understanding the feature usage and frequency, and understanding of customer's business goals and needs. *"Customers can choose one of the solutions or they can ask me to estimate both alternatives and they decide according to the estimation."*

The code below was included as a special case in this node because if the customer believes that a certain scenario should work in a certain way and he specified that to the consultant however it was implemented in a different way without communicating that change with him. Example from a consultant participant who worked on projects for the public sector: *"The users in these projects usually have the approach of: I know exactly what I am doing but I do not know what's in that system, and I want to do this task and I do not know how. This affects their feedback, as they assume that they were trying to do a normal thing but the system is not working correctly."*

5.5.2.2.8 Failure to Communicate Impact

Failure to communicate impact refers to problems related to the miscommunication of the impact of a change requested by customers on other modules (parts) in the system. *"Sometimes implementing the impact of a change is more than the change itself, and the customer does not understand why the estimate is that big. This is due to lack of information from the customer, or the analyst overlooked some aspects or was mistaken."*

Thus this information must be transferred to the customers to increase their understanding of engineer's decisions that they do not agree with or think does not support their expectations. *"The main issue is that sometimes customers request changes without being aware of the cost it imposes on us. Sometimes we agree on a certain scenario with the customer and after it is implemented he adds some extra requirement modifications thinking that this is a simple task since the feature (or scenario) is already implemented."*

5.5.2.2.9 Failure to Communicate Technical Task Dependencies

Failure to Communicate Technical Task Dependencies captures problems that result from miscommunicating technical task dependencies to customers that can sometimes result in conflicts with customers. Example from what engineer participants said: *"we have to implement base data (tables, lookups) and users first thing in the application. From the customer's view creating users is not a feature, he thinks that reporting module is more important, however, in order to do the reports, there is a technical dependency and we have to create the users first."*

These problems frequently relate to configuration and deployment problems. Definitely, customers lack this view of the system. So it is the team member's job to communicate that

with them to avoid any failures and disappointments. For example, *"a customer is requesting g a new feature in a module that has a build error, which means that the runnable module cannot be built from the source code. Technically if we implemented this feature it cannot be deployed on the customer's system. Therefore, the customer will not be able to use it unless the build problem is resolved first"*

5.5.2.2.10 Neglecting End-Users' Opinions

This node captures the cases related to the collection of feedback from a selected number of users rather than analysing and communicating from a broad range of users and roles. Priority is given to customers (as they request and pay for the system), while end users have a marginal role, despite the fact that they will ultimately experience and benefit from the system. This is almost the case in most software companies. This causes lost information due to the high level view they have and their awareness of the goals but not with the requirements and detailed workflows.

"For one of the projects, we made a system for a certain department for the government sector. Part of the system is used by them only, and other modules are used by external users. The real estate module in the system was made public and used by other users from different departments or any external user." So consultants and engineers do not deal with all these users they only deal with the customers to whom they delivered the software, which in this case are the government representatives.

"Usually, they have hotlines or other communication media, where users can report problems to them, and they report to us. This may cause problems such as *"lost information, because they communicate the problems from high level point of view, and sometimes they neglect the low level needs of the end-user who is directly using the system in his daily workflow."*

5.5.2.3 User Experience Problems

This thematic area refers to a customer's total experience in using a particular product, system or service. The first requirement for a great user experience is to meet the exact needs for the usage of a product or a service. To achieve this, users must have all necessary information for their task without hindering their experience. However, engineers do not provide usable supportive means for customers to help them understand how the system or features perform their tasks. This lack of communication, leads to lack of user awareness that causes less user acceptance.

5.5.2.3.1 Usage Scenario Unawareness

A usage scenario, or scenario for short, describes a practical example of how one or more users should interact with a system. They describe the steps, events, and/or actions which occur during the interaction. Scenarios can be very detailed, indicating exactly how someone works with the user interface, or reasonably high-level describing the critical business actions but not the indicating how they're performed. *"Problems that may occur when customers try a solution provided by engineers are that the written scenario is incorrect or unclear. So we give him feedback with the corrections (e.g. missing steps)"*

Lack of proper documentation of usage scenarios in requirements specification (See section 5.5.1.1.1), or lack of communicating them with the customer will make customers less aware of the capabilities of the software they using. This leads to less user acceptance, and more importantly will cause more time and effort in resolving and convincing customers that they reported invalid problems. *"Most customers in the beginning of their system usage report every problem they encounter as a defect and needs to be fixed. This is due to their inexperience with the systems that makes them unable to differentiate in some times between defects and help requests."* It is the consultant's job to qualify whether the issue is a valid defect or not. *"80 % of the problems are reported as defects at first."*

5.5.2.3.2 System Holistic View Unawareness

One of the problems that usually occur especially in large systems is the customers' lack of variability awareness of the system configurations and relationships between tasks or features. *"Customers report issues as bugs and they know it is not a bug, but the real problem is not usable in the way he has required or imagined."* This also means that customers are not aware of the possible existing multiple entrance points for a feature. *"Customers may need to know the roles that will use a certain feature or module to be able to recognize the correct mapping of entrance points from different parts in the software."* Awareness of all these factors could lead to mitigated customers' issues as there might be easier ways to perform a task on hand.

Also, the customers are always not aware of the impact of the changes they request on the system. *"Especially because the system is used by multiple users and each user uses a specific module (i.e. role) in their daily work so they are not fully aware of all the functionalities in the system."* For example, *"if the user wants to change a certain equation in the module he uses instead of adding, he wants it to multiply. Mostly, he is unaware that another user will be affected by this change."* The impact of the changes he requests should be clearly communicated with him to ensure he understands the workflow of handling the change request, the technical dependencies, and the priorities.

5.5.2.3.3 Feature Unawareness OR Misunderstanding

Same as Usage scenario unawareness there is the users' feature unawareness (i.e. they do not know that a feature exists), or feature functionality misunderstanding (i.e. they know the feature exists but do not know what does it do, or they are mistaken about its real purpose). *"Usually the way they report problems by simply saying the feature or scenario is not working. Sometimes the feature or scenario is actually working correctly, but the real problem is that they do not know how to use it."* Lack of user help, suggestions, training, or communication causes this level of unawareness. This leads to decreased user acceptance, and more importantly will cause more time and effort in resolving and corrections.

5.6 Confirmatory Interviews

In this section, the purpose of the confirmatory interviews, the session planning and participants' recruitment, the prepared questions and the results are all explained.

5.6.1 Purpose of the Confirmatory Interviews

Confirmatory research (Onwuegbuzie 2003) is where you have a good idea about the research topic. That is, you have a theory (or several theories), and the objective of the research is to find out if the theory is supported by the facts.

In this research, an understanding of the maintenance problems that engineers encounter during the maintenance phase, along with the different categories of missing information that cause these problems were captured. So, another confirmatory study was planned and conducted in order to review and verify the previous study results with further number of participants with different experience and backgrounds. This is to collect their opinions about the previous study results and provide insights about specific situations or gaps they discover.

For a confirmatory analysis, any deviation from the pre-specified analysis will be explained and justified. If a deviation is necessary, results from the originally planned analysis will be reported, as well as the results with the deviation. Deviations include any data transformation, adjustment, or exclusion criterion that was not pre-specified.

5.6.2 Confirmatory Interviews' Sessions and Participants

Five participants were recruited to join this study. The participants have different backgrounds, roles, and experience levels, as they work in different companies. This helps reduce any bias and/or subjectivity in the results coming from the exploratory interviews study that was conducted with participants coming from the same company. Also, their varying roles will help us verify how the lack of proper feedback acquisition and

communication affects the different phases of the evolution process, which can inform our design to the new engineering process.

Each interview lasted average 13 minutes. The total timing of the confirmatory interviews was 68 minutes. Before each interview a 30-minute introductory session was conducted with the participant to introduce the exploratory study results shown in Figures 8, and 9, and explain the purpose of the study.

Below in table 9 is the participants list that explains the experience and company background for each participant recruited in the confirmatory study:

Table 9. Participant's Experience and Company Background

Participant	Experience	Company Background
1	6 years	A female researcher who has an MSc on how to improve the bug fix time prediction models using several classification models. She has experience in using bug tools specifically Bugzilla. During her research she was exposed to the maintenance/ evolution tasks and gained knowledge about how end-users report bugs at runtime and how engineers handle them.
2	3 years	A male junior software engineer who works in a global IT Services firm providing Cloud-based and On-premise solutions with an emphasis on Advanced Analytics, Enterprise Mobility, Performance Management, and CRM. The firm has a 20-year proven track record building award-winning solutions for Telco, Banking, Manufacturing, Agriculture, and Government. The firm has global offices in several countries. They are ISO 9001:2008 and CMMi Level 3 Certified.
3	4 years	A female analyst at the Information and Documentation Centre (IDC) that was established in 1983 in an Arab League Organization that provides educational services. The main objective of IDC is to develop administrative and management information systems that help users and managers in different departments of the Academy to do their work in an easy, accurate, productive, and compact way.
4	10 years	A female participant who worked as a technical support for several projects in her company. Her career then moved towards projects' infrastructure as she worked as a system administrator. Then she worked as a DevOps engineer in several projects. She works in an Egyptian pioneer software

		house in the field of Transport Automation, Integration Solutions and Business Process Management. The software house develops local and regional ports in the field of Information Technology and Communications. Their enterprise modelling products and solutions enable transport organizations to visualize, understand, analyse, improve, audit and continually enhance complex operating processes and IT infrastructures.
5	8 years	A male participant who has an eight year experience in the field of software development. In the past 3 years he has been working in a global software house that has been firmly established in e-business for more than 20 years. They provide services and specialize in software solutions for industrial customers and public administration. His main role is a senior software engineer. However, he is involved in several tasks in the project such as: project start-up tasks like defining requirements and gap analysis. Also, in organizing requirements and estimating its timing, formulating a work breakdown structure and allocating resources for executing the tasks. He is also involved in testing tasks and deployment on customer's site.

5.6.3 Confirmatory Interviews' Questions

1) Can you tell me about your practical experience in software development and the roles you have undertaken?
2) Evolution process tasks are triggered through inputs from end users (i.e. feedback) where they report problems, or request enhancements and/or new features. Participants have informed us about the problems in users' feedback and the different kinds of missing information that could influence the interpretation of these feedbacks or affect making best use of it to inform maintenance decisions. Do you agree/ disagree with the findings? Could you say something more about that? Do you have further examples of this?
3) Participants have also pointed out on the importance of other inputs besides users' feedback that could affect the evolution tasks or maintenance decisions, and can sometimes lead to conflicts such as goals, requirements' specification, and requirements' dependencies, also historical information, and progress tracking information. Do you agree/ disagree with the findings? Could you say something more about that? Do you have further examples of this?
4) In our study the problems in the evolution tasks that are triggered by miscommunication between both the end-users and the software engineers were

also being investigated. This resulted with miscommunication categories from both sides (end user or software engineer) in which they either communicated mistakenly, unclearly, or inadequately. Do you agree/ disagree with the findings? Could you say something more about that? Do you have further examples of this?

5) Participants also discussed the problems they face due to lack of relevant information that guide them in accomplishing several change identification and evolution processes tasks starting from interpreting the users' feedback to understand the problem and trying to reproduce it followed by estimation, prioritization, and planning. Engineers and developers spend considerable effort trying to collect useful information needed from users that can help them in accomplishing these tasks both successfully and in a timely manner. Do you agree/ disagree with the findings? Could you say something more about that? Do you have further examples of this?
6) Do you have any comments, suggestions or advice about our work that you would like to share?

5.6.4 Confirmatory Interviews' Results

Participants shared their opinions on the thematic areas developed through the main exploratory interviews study. They reviewed the classification of missing information and evolution problem categories. Minor adjustments were suggested for the classification of the problem categories. They agreed on the results by sharing their own experiences to emphasize its importance, and show its high coverage to categories of problems that they are exposed to during the maintenance and support phase.

More importantly they provided many useful insights that can inform our next research steps. They discussed their expectations about the capabilities of a feedback acquisition tool that ensures valid entry, storage, and linking of feedback coming from both end-users and software team. Additionally, they emphasized the importance of utilizing RE models and its role in keeping requirements information up-to-date. Finally, they discussed the need for a staged systematic process to govern the communication between end-users and the software team members to inform the change identification and evolution process.

5.6.3.1 Opinions about Missing Information Classification

Participants agreed on the themes of missing information that was reached from our exploratory study and were described them as *"detailed and holding different perspectives affecting several roles in the maintenance/evolution process tasks"*. For example, low level information in feedback descriptions is a category that affects the analysts working on understanding and validating the end-users feedback. Requirements information affects all

different roles like analysts, software engineers, team leaders and managers too. It was also pointed that "*it's very useful to capture information such as historical information as this could help that management roles benefit from previous cases and apply them on newly occurring cases"*. Also, "*progress tracking affects the analysis and planning phases that are handled by both team leaders and managers working on current issues on hand, and it may also affect newly added bugs affecting their estimation and release panning"*.

Another participant has worked on projects developed to work on different platforms for example: mobile, web and desktop where UI is a major concern. So he completely agreed with the types of missing information in user descriptions and provided further examples to demonstrate its necessity. For example, "*in case of reporting UI problems screenshots are necessary to visualize exactly the different aspects of the problem like the dislocation of components, or the colours"*. Also, *"when working on projects running on different platforms environmental context information becomes mandatory to be able to reproduce problems in the same manner they occur to the customer"* which will enable engineers to better understand the problem on hand to later resolve it.

Additionally, another participant after reviewing the results for the classification of missing information, the engineer added that "*it is very important to attach configuration file to any change request.* She also added that *"this is the key element for resolving any issue at maintenance time*."

5.6.3.2 Opinions about Maintenance Problems Classification

Two of the participants commented on the problem themes specifically the communication problems. They said that *"it needs a further detail that differentiates the end-user related problems from the engineer related problems"*. Otherwise, all participants confirmed that the *"problem categories are detailed enough and organized in a way that relates it to the evolution process phases"*.

It was added that further benefit from this categorization could be achieved by *"relating problem categories to types of missing information that causes them, and conclude patterns of recurring problems and their causes".* This could help in devising targeted solutions or help engineers to avoid them early in future issues.

An example from another participant regarding the relation between improper requirements documentation (which is a missing information category) and the maintenance / evolution problems is that in a project he was asked to refactor a code base. When the project began, more problems were encountered regarding performance, and platform issues that affects the proper functioning of the webs application, than those that were reported by the customer in the beginning. *"The major reason for this situation is that the project's code was overwritten several times, as it moved from one developer to another who worked on the*

*same components **[without providing proper documentation]** to the final requirement or feature specification"*. This led to a state where he was working on malfunctioning components with no proper documentation. Thus, *"lots of adjustments that were not planned from the beginning were made **[adding additional milestones]**, and **[extending the deadline]** for extra 4 months over the original 2 months timing that was spent in understanding and evaluating the current situation"*.

Another example on linking problems to missing information was explained. A participant worked on a project that was developed using agile methodology. Every two weeks a new sprint began and a new requirement written by the customer was received. First task when receiving a new requirement is trying to understand it in order to be able to design and implement an appropriate solution. However, *"deadlines were not met properly due to **[missing information in the customer requests]**"*. Also, *"sometimes there was **[misleading information]** too as the **[customer depended on textual descriptions solely]**, which sometimes could be misinterpreted without providing screenshots or data files that help in completing the whole picture."*

Furthermore, one of the participants also commented on the problem **[Providing misleading feedback types]**. She said that *"she would prefer that we have also another category called providing misleading information, as the causes are different"*. For example, providing misleading feedback type may be *"due to lack of user experience so he provides a feedback as a bug while it is actually a help request"*. However, **[providing misleading information]** can be *"due the customers' inability to describe the problem properly"*. However, she said that this could be mitigated by providing screenshots or any other necessary file that could aid the engineer in understanding the problem correctly.

5.6.3.3 Insights for the Feedback Acquisition and Feedback Linking

"Users are not experienced in providing structured feedbacks and this is somehow a challenging task for them". Also, it differs from one role to another, as some roles have more technical abilities than others, and thus can provide more descriptive feedback content. So *"training the end-users and providing them with written guidelines that could further help them in performing this task is mandatory for a feedback acquisition tool"*.

Other participants described how much they value the idea of providing guidelines for the end-users on how to provide a feedback. This can be ensured by setting a default type of mandatory information for each feedback type, "*while providing descriptions as guidelines on how to use other levels of detail to provide more information and description they need"*. This can be designed by *"providing a definition for each level of detail with an example on how to use it in a feedback, which the user can use to learn how to use it in a feedback acquisition tool correctly"*.

Also, they added that in the acquisition tool or method *"it would be a nice to add a feature for end-users to link their feedback with features or components in the system"*, as this would help the engineers *"narrow down the scope and locate the problem accurately"*.

Further participants also thought highly of the idea of linking end-users' feedback together and linking them to other models whether requirements models or development models. One of them explained the *benefits of linking feedback to requirements models that could help requirements engineers identify the functional modifications in the system and its impact on other system functionalities*. Also, it could further help managers query the links *between requirements and business goals to study potential drawbacks and/or improvements for the customers' business*. Engineers also need to keep track of the requirements specifications versions with their change history, which could be used to *"examine the changing customer needs and produce better plans, and vending of new feature enhancements to the customers"*.

A participant also suggested *integrating end-user feedback with tools such as Jenkins*. Jenkins is an open-source continuous integration software tool for testing and reporting on isolated changes in a larger code base in real time. The software enables developers to find and solve defects in a code base rapidly and to automate testing of their builds. *So linking end-user feedback about feature problems can have several benefits including: faster time to market, improved customer satisfaction, better product quality, more reliable releases, and improved productivity and efficiency*.

To sum up, the continuous customer involvement through feedback acquisition will lead to better evaluation of the software, provides a base for taking better maintenance and evolution decisions, and enables engineers to support customers in an, accurate, organized, and timely manner.

5.6.3.4 Opinions about Communication Methods

Participants argued that the source of communication problems during the evolution process is due to human factors more than technology factors. A participant explained that the major problem is that there "*is no communication process that allows the software engineer to investigate with customers, and act as needed in different situations"*.

Therefore, there is a need not only to utilize the structured feedback but also to "*develop a staged process that could assist both end-users and team members in communicating together".* This staged process "*should not be in isolation from the evolution process tasks"* to enable engineers to head towards successful tasks completion through involving users as partners in the process.

As a suggestion for the next steps, a participant pointed out that she would like to see a "*walkthrough for real case of a reported bug or enhancement provided by end-users as a feedback and engineers communicating and working on resolving it"*. This would help verifying and evaluating the practical capabilities of the results. It would also help the participants give insights on possible adjustments for both the feedback acquisition tool and the communication method.

5.7 Further Study Results

In this section, the study results are explained and linked to the previous research results and the next research step. The results serve as a foundation step for a holistic approach for the acquisition, structuring and utilization of users' feedback for crowdsourced software evaluation, and provide basis for a new enhanced method for communication and change management the maintenance phase.

5.7.1 Validating Feedback Types' Components

In this study, the engineer's perspective on the role of feedback in informing their maintenance and evolution tasks and decisions was captured. As a result, we came up with a classification of sample problems and a classification of sample missing information that contributes in evolving these problems.

Another, important output of this study is validating the forums analysis results, by revising the feedback components and mapping them to the information given by the engineers. This is to confirm that they contain the needed information but still not too restrictive, which carries both perspectives.

It is important to note that the feedback types components are not only the level of details used and their methods, their definition also contains how their reference other types. However, in this study no modifications to the referencing between the feedback types was made. Mainly they were captured during the forums' analysis study due to the nature of the forums, as all feedback are reported in form of threads which makes is easily to deduce the relationships between them. In this study engineers have focused on giving us their opinion on the level of details and methods they need to make feedback types more meaningful and useful.

5.7.1.1 Topic Definition, Investigation, Elaboration Feedback Types

From the interviewees' quotes it was concluded that the mandatory information that were previously designed for the **Topic Definition** Feedback Type covers the essential information that engineers seek in a customer's feedback. Still, after analysing the interviewees' quotes, the feedback rule was expanded with the level of details that

customers provide to include extra methods of descriptions, such as snapshots as *"customers like to give snapshots to illustrate what they are saying"*. Also, a new method type named "File" was added to include files that were pointed out to be highly important such as test data files. "*For example, a user reports that an error occurred when he entered certain data. He has to provide the data that he entered, because some problems are data-specific"*

So instead, the **Investigation** feedback type was updated to enable the engineer to specify the level of detail needed from the end-user in a question format. For example, engineers may ask end-users more questions like: *"which type of import was he using, please provide the import file, which user role was used (logged in) during the import operation as it might be a problem in the user rights*." This also mandated moving the "Question" from level of details into a new method type.

Following that, **Investigation Elaboration** feedback type was restricted to contain the exact requested level of detail (i.e. the type of detail in the customers answer must be of the same as the type of detail investigated in the engineer's question). This ensures that *"end-users will have to provide exact information about the level of detail the engineer was investigating in their question to avoid end-users providing misleading information or answers."*

Additionally, **Feedback Elaboration** Feedback type means that the customer may elaborate on a feedback he has already given to *"include any type of details needed to be clarified or added in the ongoing issue because it was not included in the original post or evolved during the discussions and communication".*

Table 10 shows the rules before modifications, while Table 11 shows the feedback rules after modifications.

Table 10. Components before Modifications for Feedback Types: Topic Definition, Investigation and Elaboration

Feedback Type	Level Of Details	Method
Topic Definition	Task	Text
	Exemplification	Text
Investigation	Question	Text
Investigation Elaboration	Explanation	Text
Feedback Elaboration	Explanation	Text

Table 11. Feedback Types: Topic Definition, Investigation and Elaboration after Modifications

Feedback Type	Level Of Details	Method
Topic Definition	Task	Text
	Exemplification	File

		Text
		Snapshot
	Explanation	Text
Investigation	Level of Detail	Question
Investigation Elaboration	Level of Detail	Text
		Snapshot
		File
Feedback Elaboration	Level of Detail	Text
		Snapshot
		File

5.7.1.2 Problem Correction

From the interviewees' responses about the feedback type **Problem correction** that by definition means that a user has given an invalid usage of a feature in his problem definition. The engineers provided examples of important information that should be included in their responses in order to resolve this situation, and provide sufficient help for the end-user, which were mostly covered in our feedback type definition. The only modification made was that the explanations were expanded to include extra methods beside textual descriptions, such as providing snapshots or links to documentations that contain descriptions of the features. "*Customers link the defect to one of the requirements (i.e. use cases) that are already written in the requirements specification".* This helps the Engineers *"revise the feature's definition and usage to evaluate whether this is a valid defect or not"*

Problem Correction feedback type requires providing details about the feature definition which necessitates *"the need for proper utilization of RE models that represent features, their relations and linkage to specs to manage this response and also for keeping requirements information up to date"*.

Table 12 shows the rule before modifications, while Table 13 shows the feedback rule after modifications.

Table 12. Components before Modifications for Feedback Type: Problem Correction

Feedback Type	Level of Details	Method
Problem Correction	Feature Definition	Text
	Explanation	Text

Table 13. Modifications for Feedback Type: Problem Correction

Feedback Type	Level of Details	Method
Problem Correction	Feature Definition	Text

	Explanation	Text
		Link
		Snapshot

5.7.1.3 Mitigation: Solution, Problem Extension and Mitigation Trial Failure

From the analysed interviewees' quotes the mandatory information that should be included were deduced. First, feedback type is the **Solution** that the engineers provide was found that the rules that govern its use cover the information that engineers need which are the usage scenarios and enhanced by exemplifications *"using Snapshots, links or code snippets to help customers apply the solution"*.

Also, in the analysed problematic scenarios, the interviewees pointed that sometimes solution are inapplicable by customers. "*This is either lack of problem investigations with end-users to gather the missing information about the problem, which led to partially resolving it. Or the solution was not correct because we were not considering an aspect in their environment"* which maps to both possible feedback types from customer side which are: Problem Extension and Mitigation Trial Failure.

First, **Problem Extension** means that customers tried the solution and it resolved part of the problem but there are still some issues. From what the interviewees said that most importantly, they need "*environmental information to know the customer's configurations when the problem occurred or evolved*" and this was already covered in the feedback types rules. However, it was enhanced it to include different methods of description by including a new type of files called Configuration files, which was also verified in the confirmatory study too. Also, engineers said they need customers to *"explain the failure step and the timing of the error message"*. This can be included the level of detail Explanation (instead of trial) to include textual description and required test data if needed.

Second, case in customer responses is the **Mitigation Trial Failure** which means that the solution did not work at all, so they couldn't try to judge whether it could resolve the problem or not. This type too has to "*include the environmental context information and details about the trials they did explained by text and/or snapshot"* and enhanced it by including a new file type called the test data file to *"show they information they entered during their solution trials"*.

In conclusion, by referring to the designed rules in the forums analysis study it was found that they cover what the engineers have pointed out in the interviews to be necessary information. However, two main updates were needed which are: 1) including two file types as explained above, and 2) updating the Problem Extension and Mitigation Trial Failure Detail types (Explanation and Trials). Below in Table 14 is a summary for feedback types'

components before the modifications, while Table 15 summarizes the modifications made after examining the missing information that the interviewees explained.

Table 14. Components before Modifications for Feedback Types: Solution, Problem Extension and Mitigation Trial Failure

Feedback Type	Level Of Details	Method
Solution	Usage Scenario	Text
	Exemplification	Snapshot
		Link
		Snippet
Problem Extension	Environmental	Text
	Trial	Text
Mitigation Trial Failure	Environmental	Text
	Explanation	Text

Table 15. Modifications for Feedback Types: Solution, Problem Extension, and Mitigation Trial Failure.

Feedback Type	Level Of Details	Method
Solution	Usage Scenario	Text
	Exemplification	Snapshot
		Link
		Snippet
Problem Extension	Environmental	Text
		Snapshot
		Configuration File
	Explanation	Text
		Test Data File
Mitigation Trial Failure	Environmental	Text
		Snapshot
		Configuration File
	Trial	Text
		Snapshot
		Test Data File

5.7.1.4 Mitigation Correction

From the situations explained, the engineer pointed out the important information that should be included in the solution corrections sent to the customers, which include detailed usage scenarios and explanation of the modifications made. However, in the **Mitigation Correction** feedback type that was previously designed, only the textual explanation of the modifications made was included. So it was enhanced it to include the usage scenario as a mandatory component too, as *"it would be easier for customers to apply the new solution*

scenarios rather than depending only on textual descriptions of modifications to solution that already failed".

Furthermore, they pointed out that the reasons why Solution errors could occur is that there was *"missing information in customer's feedback that were not properly investigated"* in the Acquisition and Analysis phase. After the customer provides a problem extension or Mitigation trial failure, the investigation starts over to gather any needed missing information.

To illustrate the modifications that were made, below in Table 16 is a summary for feedback type's components before the modifications, while Table 17 summarizes the modifications made. Mainly, the modifications made were tailored to *"represent mitigation corrections in a similar way to which solutions are provided to customers so it could be used as an updated version of a working solution and can be further used as a reference for updating the requirements/features specifications."* (i.e. using the same level of details and methods).

Table 16 shows the rule before modifications, while Table 17 shows the feedback rule after modifications.

Table 16. Components before Modifications for Feedback Type: Mitigation Correction

Feedback Type	Level Of Details	Method
Mitigation Correction	Explanation	Text

Table 17. Modifications for Feedback Type: Mitigation Correction.

Feedback Type	Level Of Details	Method
Mitigation Correction	Usage Scenario	Text
		Snapshot
		Link
		Snippet
	Explanation	Text

5.7.2 Utilization of the study Results towards the Next Steps

Analysing the situations that engineers provided to capture the problems and their related missing information in this study (both the exploratory phase and the confirmatory phase) served in four different purposes:

1) To determine the types of missing information in users' feedback that are most important to engineers, and comparing them to the novel classification of feedback types and their components that was developed in the forums analysis study. This helped validate the developed classification and ensure it carries both perspectives as it was developed from users' perspective and improved from engineer's

perspective without requiring extra information from end-users, and hindering their experience;

2) To determine the types of problems that engineers encounter and their causes. From the analysis results, an emphasis on the communication problems was made. This serves two purposes: a) highlighting the need to develop a communication process to guide both engineers and end-users in their interaction. This will contain the possible variations of situations they may encounter which can be summarized from the situations they explained; b) utilizing the new list of updated feedback types in the communication process to be used as a tool by both the end-users and engineers to provide structured feedbacks that could be linked to features, requirements, and goals.
3) It helped in endorsing the need to employ requirements models and use them as a backbone to link all the feedback to maintenance and evolution tasks to keep the requirements' information up-to-date and give a holistic evaluation view of the system, because it will enable engineers to view two different levels of evaluation: a) evaluating the features in use and relating problems to them and thus by simple measurements it can be known which features are more important to users, which features are more problematic, and how far does a problematic feature affect other features, or tasks; b) evaluating the goals of the system, because by relating the feedback to customer goals it can be systematically concluded which goals are unfulfilled, and would cause stakeholders' dissatisfaction. Thus, this linking would help engineers in taking accurate and efficient decisions.
4) An important input to the next research study, which uses the participatory design approach, is the scenarios that the participants will immerse in. This study was full of examples of situations that engineers encountered with end-users in different phases of the evolution process and their causes. This provided insights that will help in designing the required scenarios for our next study.

   The immersion scenarios will be used to engage the engineer and end-user participants in fictional situations (Buskermolen and Terken 2012). The motivation for using fictional scenarios that are based on real situations in participatory design is to invite participants in design to re-think existing practices and imagine what their practices might be like if established conventions were altered.

5.8 Threats to Validity

This study has three main threats to validity:

1) The first threat is one of the common issues when designing an interview and relates to ensure whether the questions were understood by all participants as planned. This threat was addressed through a pilot test that was conducted on a typical participant (software engineer) then some questions were revised and

modified to ensure clarity as explained in section 5.4.4. The results of this pilot interview were not included in the study results.

2) Another important thread to validity when conducting interviews specifically, unstructured interviews is eliciting behaviour or information from the participants that will be consistent with the researcher's expectations. This may occur, because the interviewer is not obligated to follow a standardized script that is used with all interviewees, the interviewer is free to adapt the questions asked, thus creating conditions to confirm his or her expectations.

   This threat was reduced, as in this research, semi-structured interviews were conducted with a standard set of questions that will be asked to all participants and also another set of questions that may be asked to elaborate on certain issues to cater for the different roles being interviewed. The roles were decided in the design of this study according to the criteria of whether their job entails communicating and handling issues with customers and/or end-users during the maintenance phase.

3) A further typical threat relates to participants' selection. That is the researcher who recruits the candidates may be the same individual who interviews them and who makes the final participant appointing decision. The reason why validity is threatened if the process is not properly controlled is that the interviewer who has seen the participant's application material, or has been given information about the candidate creates an implicit expectation, which he then seek to confirm.

   The selection process was controlled by separating the selection function, which was performed by the academic advisor and the help of fellow researchers who did not take part of this study.

5.9 Summary

In this chapter, the interviews study with engineers was explained. The aim of the study was to explore the utilization of the feedback structures developed in the forums analysis study to accomplish the evolution tasks. The study resulted in two thematic maps of concepts related to the classification of problems that engineers encounter in the maintenance phase and their causing problems. These results were the main driver to the study explained in the next chapter to further evolve the initial designs resulting from the employment of the previous results, and to confirm their usage in practice with actual end-users and engineer.

6. Designing a Method for Feedback Acquisition, Communication, and Requirements Updating

As discussed in the previous study (i.e. interviews study) with the engineers that the lack of structure in the users' feedback, leads the engineers to spend much time and effort in communication with users in order to interpret feedback, obtain missing information, analyze problems and plan the changes. Moreover, the lack of formalism and systematic approaches to extract requirements changes makes it hard to keep requirements information up to date along the evolution process.

In this chapter, the purpose of the Participatory Design study (Spinuzzi 2005) is explained and the results reached from it. This study aimed at: exploring the practical use of structured feedback developed earlier in this research in the communication between end-users and engineers; also, evaluating the use of existing requirements engineering models and its role in representing requirements information to facilitate the engineers' tasks; furthermore, devising a new process for feedback acquisition and communication between users and engineers, which utilizes structured feedback and provide guidelines for a more methodical communication; and finally, extending this new process for extracting requirements information, updating feature models and feature specification documents for more accurate usage when new issues arise along the maintenance phase.

This chapter is organized as follows:

- Section 6.1 the research method is explained, the purpose of the study, the software that was employed in the fictional immersion scenarios developed and the participants recruited.
- Section 6.2 explains the two types of sessions that were conducted. First, the introductory session purpose and handling approach is explained. Second, the design sessions preparation is explained in full details with all the materials and tools that were prepared.
- Section 6.3 explained the study results, which is divided to three main parts. The first part related to the explanation of how the new process for acquisition and communication evolved and concluded. The second part is mainly related to the utilization of feedback types and the updates that occurred to their structures. Another concern was discussed related to the formalization of textual descriptions through the use of controlled English. Third, the new process for acquisition and communication was further extended to include the utilization of the gathered feedback to extract information for updating the software's feature model, and the new developed structure for feature specification document.

6.1 Research Method

In this section the adopted research method will be discussed and the motivation behind choosing it will be justified. The purpose of the study will be stated. Also, the software that was employed in the design of the immersion scenarios that was used in the design sessions will be discussed and the goal behind utilizing this kind of software in the study will be argued. Furthermore, a summary of the participants recruited in the sessions and their backgrounds will be described.

6.1.1 Participatory Design Method

Researchers can create tools, products or services with lots of brilliant capabilities, but sometimes are neither usable nor understandable for end-users which lead to its misuse and/or bad user experience. To overcome this problem many user centered approaches were developed. However, user centered approaches suggests that users are taken as centers in the design process, consulting with users heavily, but not allowing users to make the decisions, nor empowering users with the tools that the experts use (Johnson and Hyysalo 2012).

On the other hand, Participatory Design (Kensing and Blomberg 1998, Spinuzzi 2005) (PD) labels inventive activities that are done with end users in order to explore their ideas about a tool, product or service to ensure that it meets their needs and expectations (Kanstrup 2012). By this the end product is developed hand in hand with the actual intended audience and thus leads to better results and experiences, as the method can give clear insight into their vocabulary, priorities, and the things they value.

In this research several studies using different data sources were conducted, different collection methods and different design to come up with feedback concepts, and rules, evolution problem categories and their causes. Using the participatory design approach will facilitate the validation of data through cross verification from different sources; the previous results will be employed to study the same concepts in practice.

This triangulation is a powerful technique (Jick 1979). In particular, it refers to the application and combination of several research methods in the study of the same phenomenon. From the advantages of using this technique: 1) it can be used in qualitative studies for both validation and inquiry; 2) it is a method-appropriate approach of establishing the credibility of qualitative analyses; 3) by combining multiple observers, concepts, methods, and experimental materials, it is expected to overcome the weakness or biases and the problems that come from single method, single-observer studies.

Also, in the previous studies that were conducted in this research, the techniques used are: focus groups, forums analysis, and interviews, which are not immersive enough for

participants to give practical solutions, as it does not allow them to speculate what the solution would look like. This is another purpose why it is desired to use the participatory design method so that participants could engage in the problem to provide better solutions. This can be achieved by giving them initial prototypes or mock-ups (Clement, McPhail et al. 2012)of the solution to help them visualize the idea and then provoke them with specific requests related to the topic.

Finally, participatory design will allow us to get in touch with how interfacing would look like, not in the sense of graphical user interfaces and visual concepts, but more in the sense of how they would like this to be presented in form of steps and workflow, which will further help direct the interfaces according to their inputs. All these dynamics were hard to capture during interviews. Therefore, the participatory design method will help us come with an enhanced design of how the solution should look like from user perspective.

6.1.2 Purpose of the Study

The purpose of the study was:

1) To design and evaluate the use and benefits of the feedback types in practice
2) To design and evaluate the use and benefits of Attempto Controlled English in structuring the end-user feedback
3) To evaluate the use of RE models and its role in facilitating the engineer tasks and keeping the requirements information up to date
4) To evaluate the new augmented software evolution process and user involvement practice in the process
5) To design an engineering process for software engineers to use in order to ensure that requirements are kept up-to-date.

6.1.3 Software Employed

The software that was chosen to be utilized for the study is Moodle (Dougiamas and Taylor 2003), which is a learning platform designed to provide educators, administrators and learners with a single integrated system to create personalized learning environments. Moodle users can use it to create courses online, upload materials, manage authentication and enrolment of courses, and other collaborative features and activities such as grading and giving comments on uploaded files and assignments.

To explore the design of the intended method the participants need to immerse in fictional problems (Buskermolen and Terken 2012) of software already in use (i.e. Moodle). The problematic scenarios that were developed for the study employed the software features as an example. Also, the software enabled the development of illustrations of domain models (i.e. feature, and business process models) for participants to explore, and work on.

Moreover, these scenarios helped the moderator control the flow of the design sessions until the session's aim was achieved.

6.1.4 Participants Recruited

Two types of targeted participants were recruited for the study: the end-users (i.e. customers) and the engineers. The plan was to produce a product that will help each on his side of communication. This is a novel practice, where end-users collaborate and work as evaluators to the software besides their roles as main stakeholders and source of requirements. Thus, it is needed to be known from their point of view how they expect such acquisition method and engineering tool to help them in providing such essential input, and also how engineers view it as an enhanced method and process (i.e. engineering task).

To conduct the sessions 10 participants were recruited 5 end-users and 5 software engineers. The end-users participants recruited all work in the educational domain where their work requires them to use learning management systems (LMS) to handle all aspects of the learning process. Since it is intended to use such type of software in the study to provide fictional scenarios that will be used for immersion in problematic situations, therefore, they were best suited for participation as they can easily understand and communicate the problems in this domain, because they are aware of it through their daily work.

Furthermore, the 5 software engineers recruited were from 3 different software companies. One of the companies is an international company based in Germany, while the other two companies at national software houses based in Egypt that provide solutions not only to customers in Egypt but also in the middle east. They are all familiar with learning management systems' features, tasks, and environment as they have previously developed LMS for different universities.

6.2 Sessions' Plan

First, 2 introductory sessions were held, where all the 5 end-users were combined in an initial introductory session, and all the 5 engineers were combined in another introductory session, to introduce the context of interest, the aim of the study, and setting the stage for discussion. Each introductory session lasted 2 hours.

Then, both types of participants were paired producing 5 pairs of end-user and engineers. Each pair of participants engaged in a design session separately to fully explore the concepts and concept design resulting in 5 design sessions that lasted in total 9 hours and 8 minutes. This acknowledges that knowledge comes in many voices, and from this perspective it equalizes participants in the design process as different contributing, each

with their difference. The total sessions conducted were 7 sessions (2 introductory and 5 design sessions).

6.2.1 Introductory Sessions

In this section it will be described how the introductory session where conducted. These sessions were conducted in order to introduce the need and purpose for the participatory design study. This was achieved by introducing to the participants the research problem, the adopted concepts from existing literature and the results reached so far from this research, both that will be utilized in the study. The purpose of the study was explicitly stated and explained. And finally, the tools that will be used by the participants during the sessions were explained and exemplified through a case study.

6.2.1.1 Concepts Introduced

In each introductory session an introduction of the research problem was made, summarized in: the issues in the current methods of acquiring users' input, and feedback analysis techniques. Also, the challenges that engineers encounter while communicating with end-users to gather useful information, and ensuring that requirements keep pace with the changing contexts they operate within were discussed.

Additionally, the adopted concepts that contributed towards building the mockups and initial version of the feedback acquisition and communication workflow were explained. Some of these concepts were developed earlier in our research such as the classification of feedback types (Sherief, Abdelmoez et al. 2015), and the rules that govern their usage, which will be used by both end-users and engineers to provide structured feedback. The classification of feedback types, levels of detail used to describe each feedback type, and the methods of description were all explained to the participants. The rules of Attempto Controlled English were introduced and the potential of their usage were explained (Fuchs, Kaljurand et al. 2006). It was also pointed out that these definitions and usages are all subject to modifications in the design sessions after being utilized in practice by the participants.

Moreover, the traditional evolution process (Sommerville 2006) for managing software changes that normally engineers go through during maintenance to handle any change type reported by the customers was revisited. This software evolution process as seen in Figure 14 does not cater for any communication between end-users and engineers and therefore communication is left as an improvised task depending on the engineer's capabilities and experience with no regulations to ensure how and when information should be gathered. When such information is collected and handled in an inefficient manner, this is one of the main causes why keeping requirements information up-to-date is a challenging task.

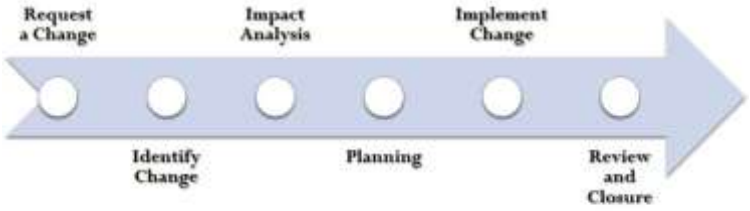

Figure 14. A simplified overall view of the change identification and evolution tasks process

Furthermore, as concluded from the literature and as gathered from the interviews study, requirements are always gathered carefully in the initial phases of the software process (Kotonya and Sommerville 1998). “Gathered carefully” means that there exist methods and tools for helping the requirements engineers in the requirements elicitation, testing and organization. However, keeping requirements up-to-date along the software process and specifically in its last phase of maintenance and support is still a challenging task. In this research it is argued that feedback while the software is in use is the main source of requirements during the maintenance phase and therefore, devising new mechanisms for gathering feedback properly could lead to better communication and also documentation of these requirements.

Also, the problem of gathering requirements information during the maintenance phase does not exist only due to that lack of guidance of how and when information should be gathered, but also due to the lack of guidance and tools on the kind of information that should be stored about feature specification after changes. Figure 15 shows a sample feature specification template (Robbins 2004) that was presented to the engineers in order to illustrate the way of documenting features. It shows by example that documenting the feature changes descriptions is entirely left to the engineer’s judgement about the suitable information needed for the documentation and means of describing them, which may hold a great deal of bias and subjectivity, due to lost information that wasn’t gathered during the communication.

F-02: FEATURE NAME	
Priority:	Essential | Expected | Desired | Optional
Effort:	Months | Weeks | Days | Hours
Risk:	Dangerous | 3-Risks | 2-Risks | 1-Risk | Safe
Functional area(s):	WORD, WORD, WORD
Use case(s):	UC-01
Description:	1-4 PARAGRAPHS. USE BULLETS OR TABLES TO ORGANIZE INFORMATION. Precise Details: • LOGICAL CONSTRAINT • LOGICAL CONSTRAINT
Notes and Questions:	• NOTE • QUESTION

Figure 15. Sample Feature Specification Document (Robbins 2004)

Finally, the concepts and usage of two types of requirements engineering models were introduced: the feature model (Kang, Kim et al. 1998, Batory 2005) and the business process model (Group 2006). It is argued that incorporating them could help produce better results in the impact analysis tasks made by engineers when handling any change request. It is claimed that incorporating RE models within the process will also help them narrow down the change scope, identify impact on other requirements/features, and thus produce more accurate estimates and better results.

In order to better introduce these concepts and help the participants get a better idea on how they will be involved during the design session, a sample case study was demonstrated. This case study used a fictional scenario (Buskermolen and Terken 2012) to present a problematic situation in the Moodle software. Also, a brief introduction about the Moodle software and its main features and capabilities was given. Feedback templates representing each feedback type and its component(s) were designed for the design sessions. They were also explained in the introductory session, and utilized for the sample case study to provide an example of how it will be used to provide structured feedback. Each template contained toolboxes for the levels of detail and Attempto for the participants to utilize them during the design session if any updates in the feedback templates are needed.

6.2.1.2 The Introductory Case Study

In this section the first scenario that was used in the introductory sessions is explained. This scenario is based in features that reside in the **Courses** module (Moodle 2016). Courses are the spaces on Moodle where teachers add learning materials and activities for their students. Courses may be created by admins, course creators or managers. Teachers can then add the content and reorganize them according to their own needs.

The courses module (Moodle 2016) contains a submodule named **Adding a New Course**. This submodule contains a group of features as shown in the feature model of this submodule in Figure 31. In this scenario the feature named **Bulk Course Creation** is utilized. This feature enables users to create several courses at once by defining them in a CSV file. For full details on how to bulk create courses, see submodule: **Upload Courses (Moodle 2016)**. This submodule belongs to the module **Courses**, and it contains a feature named **Upload Courses** which is explained in the following steps and relates to Figure 16:

- Go to Administration > Site administration > Courses > Upload courses
- Either drag and drop the CSV file or click the 'Choose a file' button and select the file in the file picker

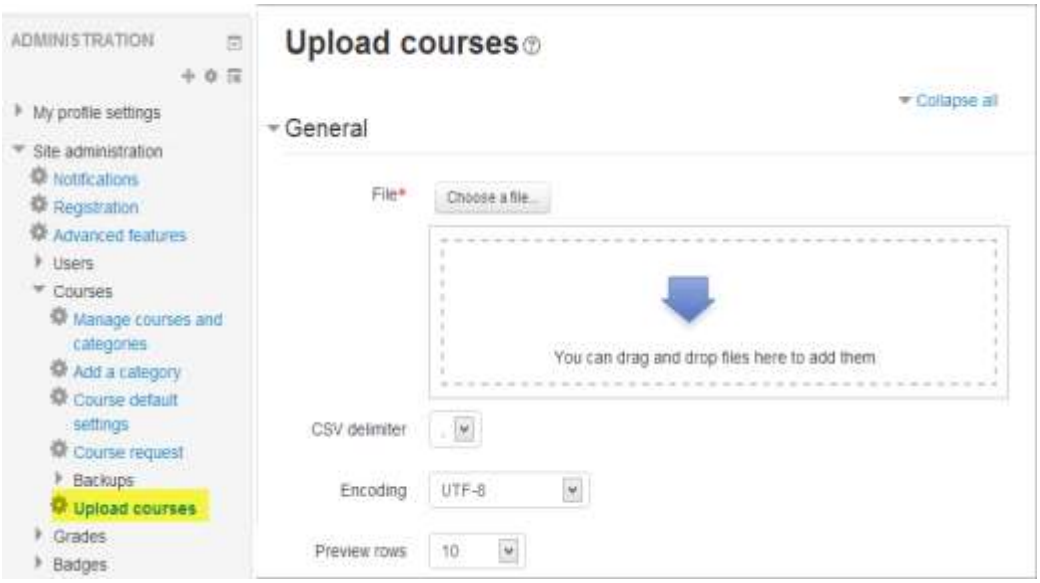

Figure 16. A Moodle Screen for Upload a Course's CSV file (Moodle 2016)

- Select appropriate import options carefully, and then click the preview button.

The Problem statement for the scenario is: suppose you (i.e. the end-user) created a CSV file containing a list of new courses you want to add. The course names are written in German language as shown in Figure 17 below:

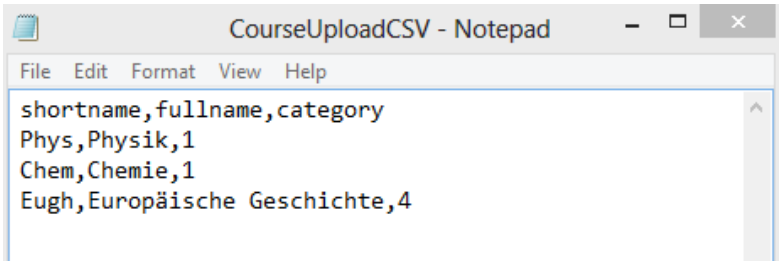

Figure 17. A Sample CSV file for Course Upload on Moodle

After you uploaded the CSV file, and you viewed the uploaded course results, the special German character “ä” was replaced with incorrect symbols “?”, such as in line 3 shown in Figure 18 below:

Figure 18. A Sample Upload Courses Results Screen on Moodle (amended from Moodle 2016)

6.2.1.3 Feedback Templates for the Introductory Case Study

The templates mock-ups (Clement, McPhail et al. 2012) were presented to the participants to illustrate: how they are expected to provide feedback, the utilization of toolboxes (ACE, Depth, and Context) to provide structured feedbacks, and to demonstrate the undertaken path in the communication workflow. This would give them an idea of how they would be involved during the PD design session.

The first template introduced was the **Topic Definition** that was used to report the problem, where an explanation to the problem was provided, and the task that was being performed when the problem occurred was reported, and a snapshot of the CSV file was provided as shown in Figure 19. This feedback was linked to a specific process and activity in the business process model and a certain feature in the feature model to further help engineers in the analysis phase, which will be investigated with engineers during the design sessions.

The second template in this scenario was an investigation template to show the participants how missing information is **Investigated** in a structured manner and linked to the initial feedback provided by the end-user. In this template two questions were asked about thee encoding option that was used and a snapshot of the results was requested as shown in Figure 20.

The third template in the thread was an **Investigation Elaboration** template in which the user is supposed to provide answers with the same depth and context types that were asked by the engineer in the Investigation template. This is to make sure that the user provides the mandatory missing information, which in our case is the answer to the

encoding option used with the created CSV file and an attachment of the snapshot of the results in Figure 18. This template is shown in Figure 21.

Finally, by this point a reply with the interpretation of the problem was provided, which in this case was a **Problem Correction**, meaning that the problem was invalid (i.e. it is not a bug), instead it is lack of user experience. This was explained by providing a feature definition that clarifies the correct encoding option to be used when writing a CSV file in German language. Also, an explanation of the steps that should be followed to perform the task and a snapshot of the form where the user should perform the task from was provided to complete the task description and prevent any confusion as shown in Figure 22.

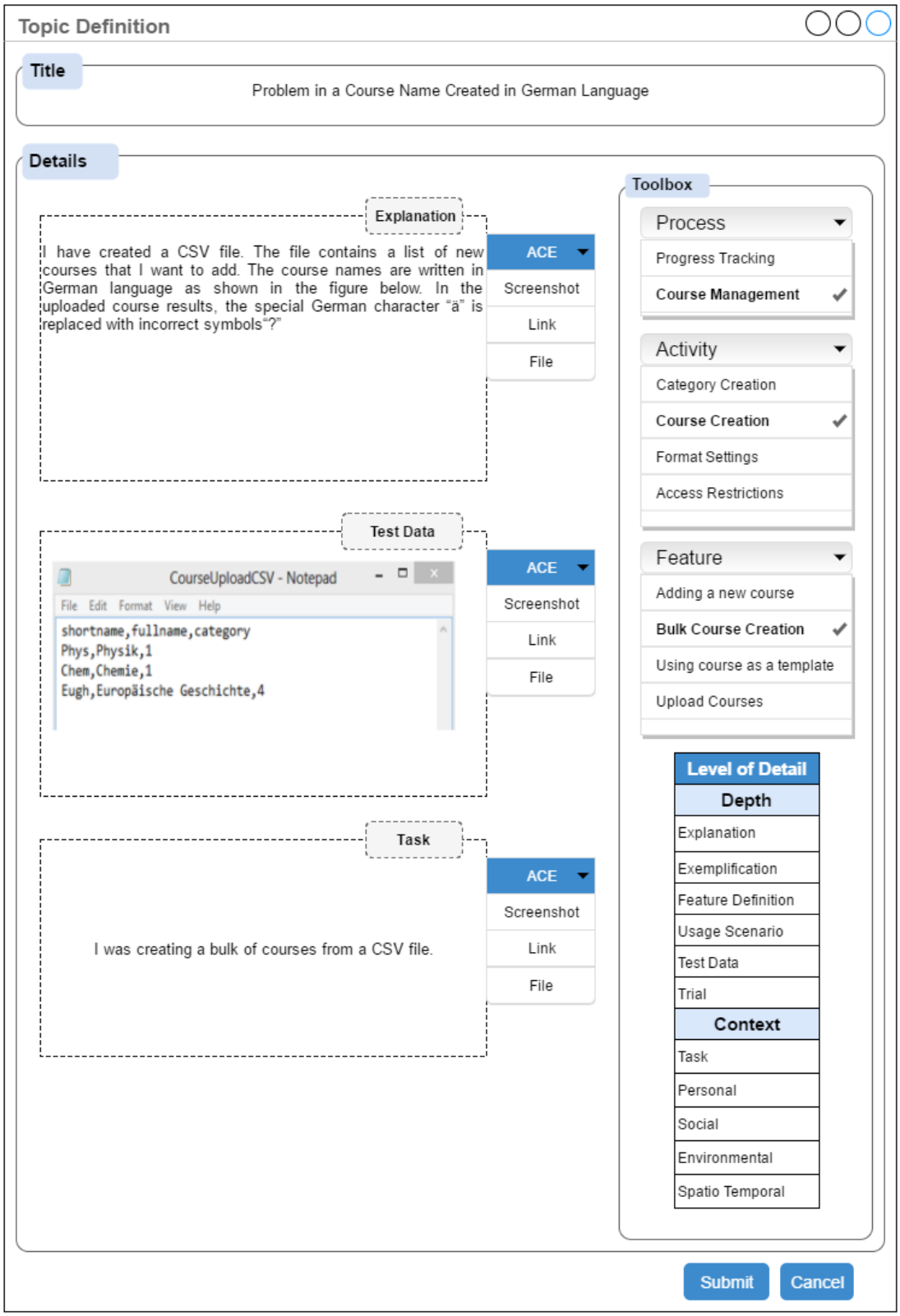

Figure 19. The Topic Definition Template Designed for the Introductory Session Case Study

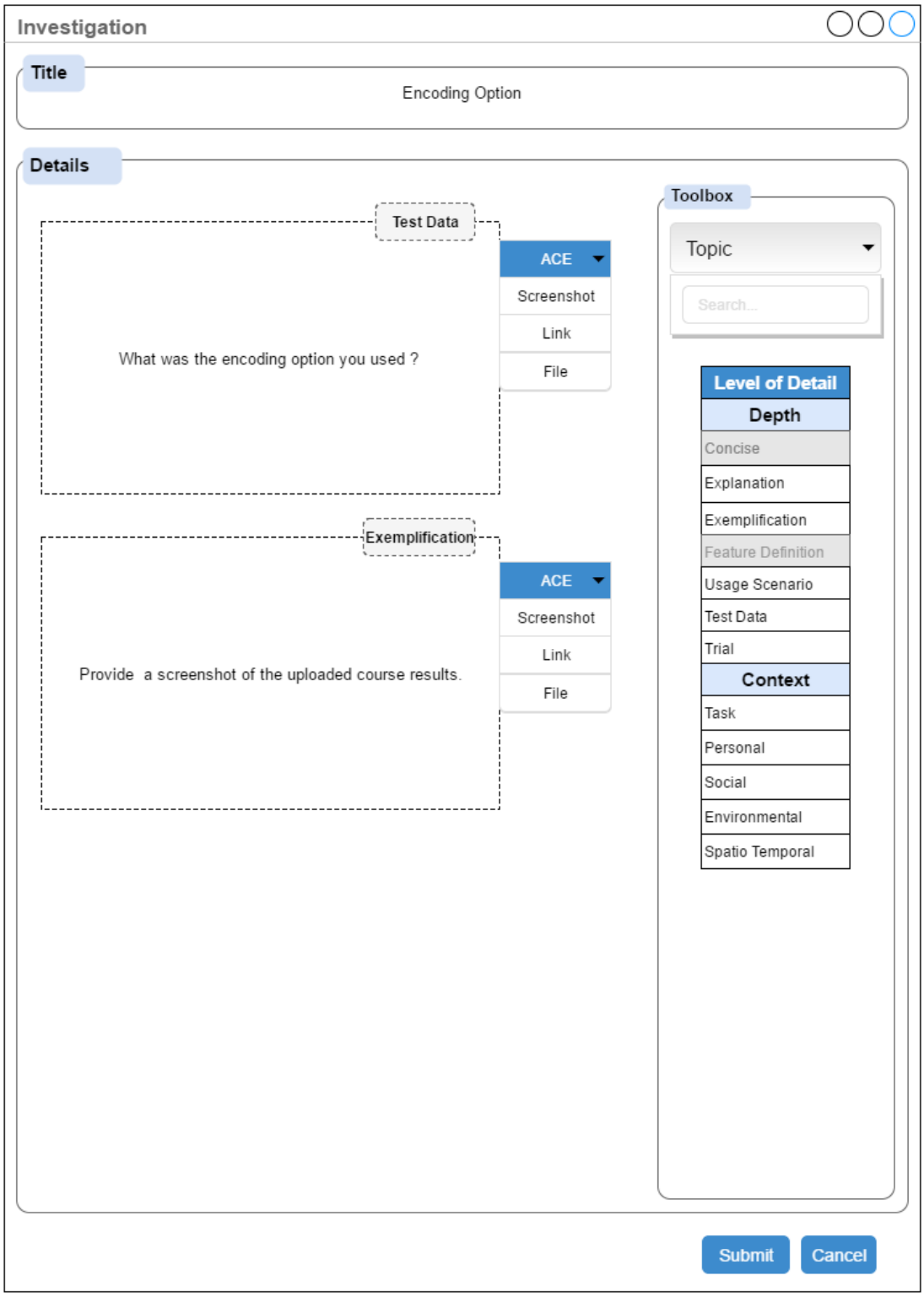

Figure 20. The Investigation Template Designed for the Introductory Session Case Study

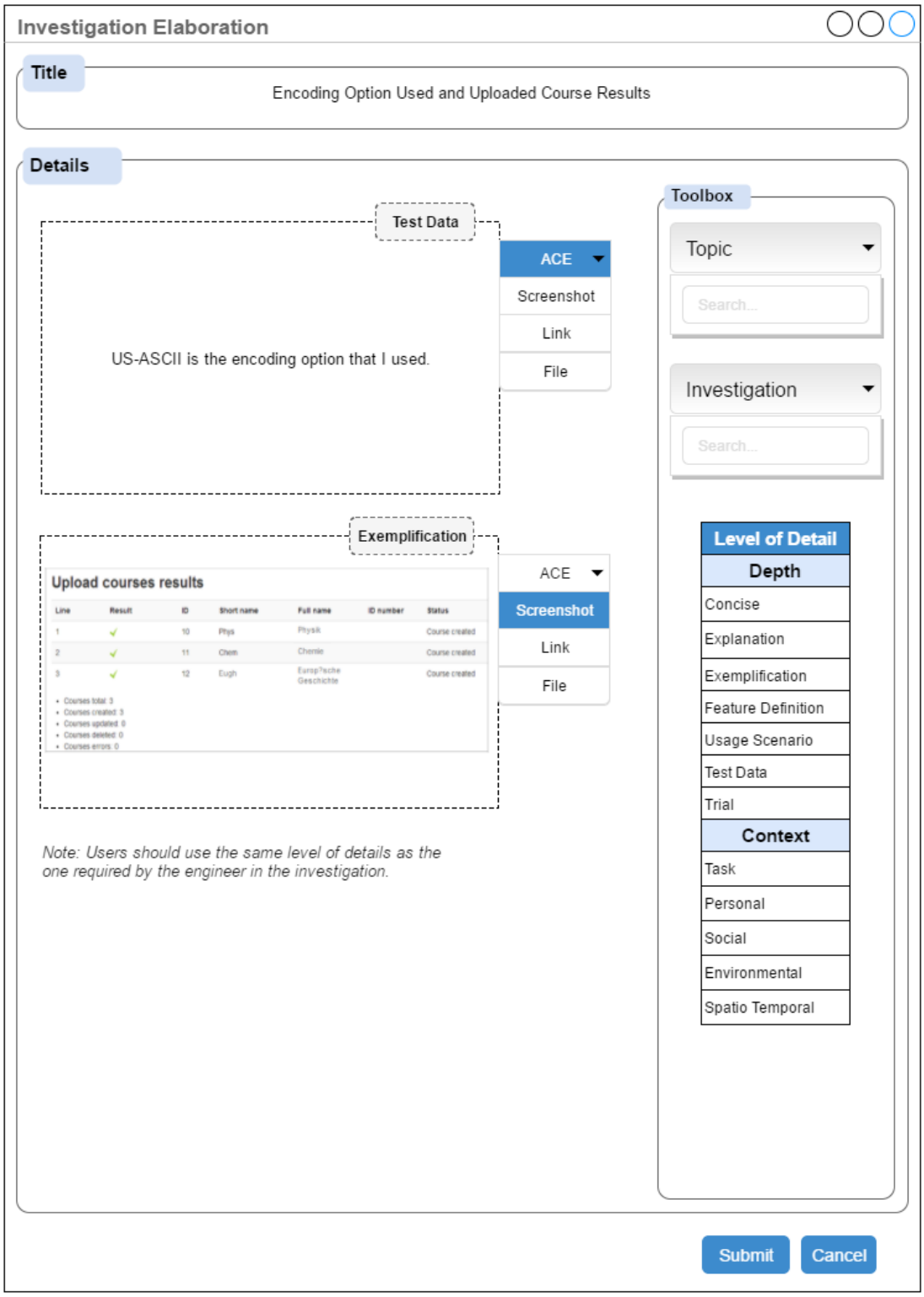

Figure 21. The Investigation Elaboration Template Designed for the Introductory Session Case Study

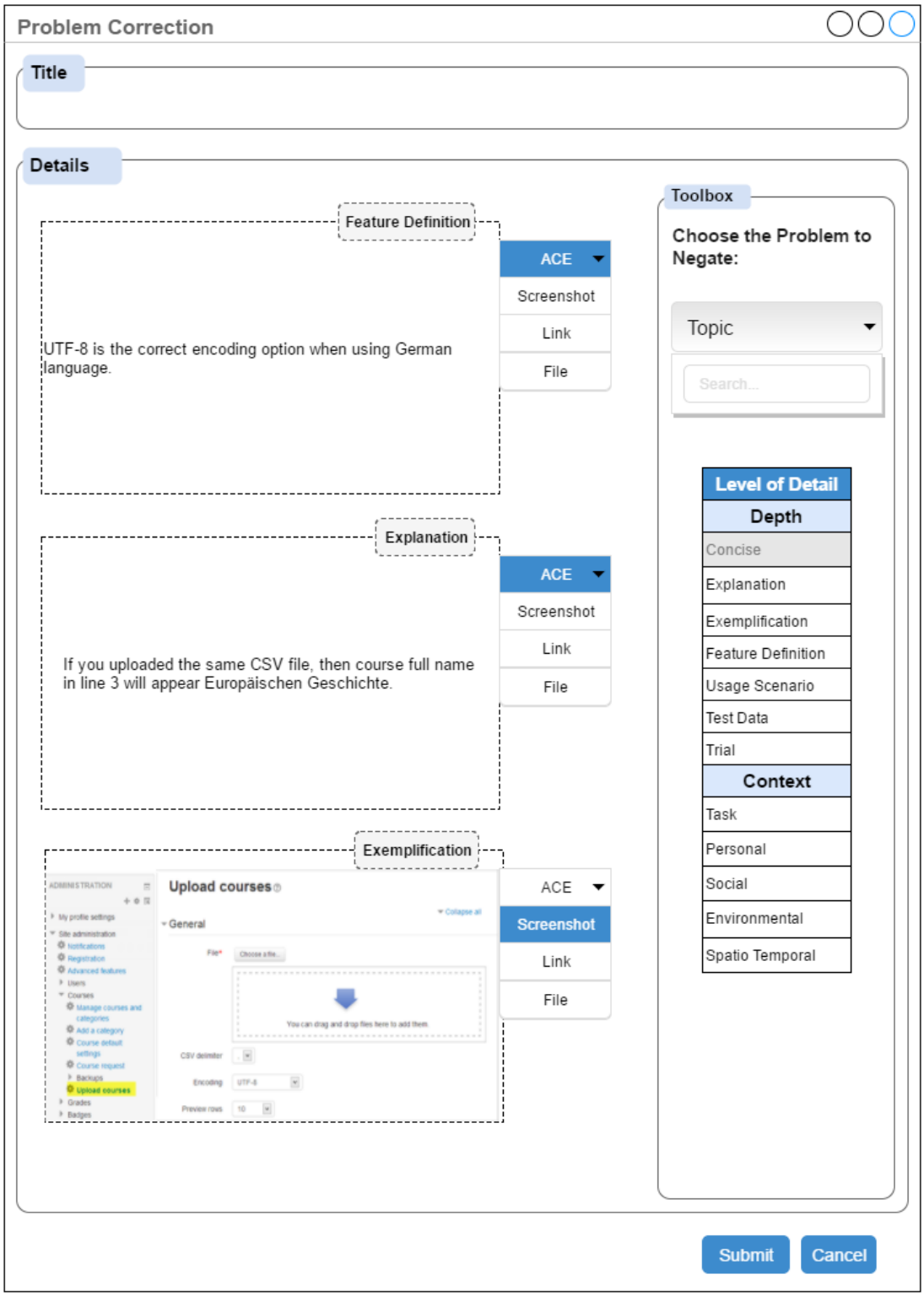

Figure 22. The Problem Correction Template Designed for the Introductory Session Case Study

Normally according to our initial design to the feedback acquisition and communication work flow, when the engineer identifies that the topic definition (i.e. the reported problem) provided by the end-user is invalid, he writes a problem correction explaining the correct definition and the communication ends.

6.2.2 Design Sessions

Each design session started by distributing the participant information sheet to both: the engineer, and the end-user. The sheet explains information about the study, how will the participants be involved, what kind of information will be sought from them, what are the advantages and the risks of being involved, will they be recorded and how the records are going to be used. All this information was explained in the introductory session, after explaining the adopted concepts and purpose of the study were explained and before explaining the sample case study. Next, in the design session each participant was given a consent sheet to sign as an agreement from them on participating the session. Each consent form was signed off by both the participant and the moderator.

6.2.2.1 Immersion Scenarios

Scenarios are “stories about people and/or their activities”. Scenarios can be presented in text, story-boards, video mock-ups, scripted prototypes etc. They support envisioning future work situations to allow the users to experience how emerging designs may affect the work practice rather than relying on the seemingly esoteric language of software developers. Using scenarios in participatory design (Buskermolen and Terken 2012) allow discussions of contexts, needs and requirements and are often the first step in establishing stakeholder requirements. They can also serve as a communication tool between different stakeholders with different backgrounds in the design session. Being selective pays off; it is better to work with a number of scenarios that are very specific than with a few that are general. Furthermore, open-ended scenarios are good to use early in the design process, while more closed scenarios may serve better later in the process when, for example, testing a particular solution.

After having the participants’ consent, the moderator distributed on both participants (the end-user and the engineer) a fictional scenario, which explained a task that the end-user should imagine he was doing on the Moodle software. The features used to perform this task were stated and their usage was explained in the form of a usage scenario as described in the following sections “Scenario 1” and “Scenario 2” and snapshots of the Moodle screens that are available in the online documentation of the software. Then a statement that explained the problematic situation that ought to occur while performing this task using these features was described. This helped the participants immerse in situations similar to real situations they encounter in their daily work, and visualize the problems as if they were real. Additionally, the engineers had an extra section in the distributed scenario, which is a suggestion of the mitigation to the given situation. This is to help them understand what should be reached from the discussion with the end-users, and also to save time thinking in an actual solution to the problem. Instead focus on how it should be written in a structured and meaningful manner so that customer can understand it easily.

The scenarios used in the design session were different than the one explained in the case study of the introductory session. In software maintenance end-users may report invalid problems, help requests, bug fixes, enhancements or new features. The two problematic scenarios that were designed for the design session were closed ended scenarios that are supposed to lead to an enhancement in the system. This type was specifically chosen, because it was intended to immerse both end-users and engineers in multifaceted scenarios that could help them explore different flows of communication. Depending on how the problem was triggered from the end-user, the communication workflow took place. The sessions' workflows will be explained in section 6.3.1.1 to show how the design evolved to its final version. All communication was held in the form of feedback going between end-users and engineers. Hence, the designed scenarios helped us validate each feedback type used and its components, and also to practice, explore and evaluate the communication workflow with its intended audience each from his perspective.

Scenario 1

In this section the first scenario that was used in the PD design sessions is described. This scenario is based on features that reside in the **Courses** module that was already introduced in the introductory session. The courses module contains a submodule named **Adding a New Course**. This submodule contains a group of features as shown in the feature model of this submodule in Figure 31. In this scenario the feature named **Using an Existing Course as a Template** is utilized (Moodle 2016). To use this feature first, create or locate on your Moodle the course you wish to use as a template and make a note of its short name. A template course might for example have common headings or section summaries or policy agreements used throughout the site for consistency. Note that, only the content of the template course can be restored but not its settings. Settings are added separately.

In order to use this feature, the user should follow the following steps:

- Create your csv file. See **Upload Courses** for accepted fields. (This was shown by example in the introductory session case study)
- From Site administration>Administration>Courses>Upload courses, add your file as shown in Figure 16.
- Preview it and scroll down to 'Course process'. In the box 'Restore from this course after upload', add the short name of your template course as shown in Figure 23.
- If you are creating several courses using the same template and you wish them all to have the same settings, you can specify these in the Default course values.
- Upload your csv file.

Figure 23. A Moodle Screen for Creating a Course from an Existing Template (Moodle 2016)

As specified in the feature above that it only copies the content and not the format settings. However, if the user wishes to set the course format this could be done through a separate submodule named **Course Settings** (Moodle 2016) that resides in the **Course** module as illustrated in the feature model in Figure 32 using a feature named **Course Format**.

In order to use this feature, the user should use the following settings as shown in Figure 24 below (These settings are collapsed by default):

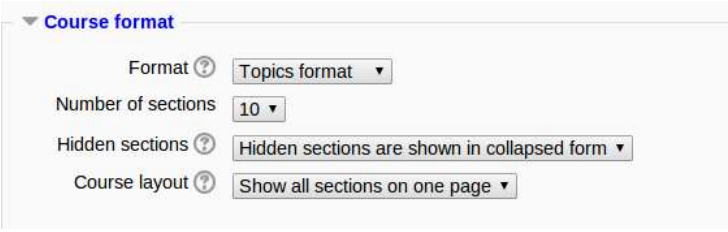

Figure 24. A Moodle Screen for Setting a Course Format (Moodle 2016)

The Problem statement for this scenario is: Suppose the user used the feature *"Adding a new course> Course Templates> Using an Existing Course as a Template"* to create a new course. However, as mentioned in the feature description, it only copies the content of the course and not the format settings. Suppose the user wants to add the same capability for the "Course Format" feature to enable him to use the same format of the template course that was used, while creating the new course. Write a feedback reporting that enhancement to the development team.

As mentioned that beside the scenario documentation explained above a description of the suggested solution to aid the engineer was also provided. The solution description could be used when the communication between the end-user and the engineer reaches the solution stage where the engineer should provide the Mitigation Feedback type.

The suggested solution in this situation will need an update in the **Using an Existing Course as a Template** feature that must save a reference of the template course in the new course to be used in further functionalities. Also, another update will be needed in the *Course Settings* > *Course Format* section. That is to add a new button such as in Figure 25 below that will be enabled only if the user used the course templates in the creation.

Figure 25. The Modified Moodle Screen for Setting a Course Format (amended from Moodle 2016)

Scenario 2

In this section the second scenario that was used in the PD design sessions is described. This scenario is based on several related features that reside in separate modules. The summary of this scenario is to assign users to certain roles in a course category. In order to do so, these roles have to be enabled as assignable. To enable such assignments the user has to define the context in which he is permitted to assign users to roles, which in this scenario is Category context. For dependencies between the features used in this scenario see the feature models in Figures 20, 21, and 22.

The first feature is **Assigning Users a Role in a Course Category** that exists in the **Course Category** submodule part of the main module **Courses** (Moodle 2016). Course categories organize courses for all Moodle site participants. The default course category on a new Moodle site is "Miscellaneous". A Course creator, Administrator or Manager can put all courses in the miscellaneous category. However, teachers and students will find it easier to find their classes if they are organized in descriptive categories. The list of courses within a category by default shows the teachers and the summary of each course. If the number of courses within a course category exceeds 9, a short list is shown without teachers and summary.

When using the feature **Assigning Users a Role in a Course Category**, by default it first directs users to the module **Enrollments**, which is responsible for the process of assigning users to roles in a course. The submodule **Category Enrollments** (Moodle 2016) is responsible for allowing users to be enrolled in all courses within a category.

To use **Category Enrollments**, it must be enabled by an administrator in *Site administration* > *Plugins* > *Enrollments* > *Manage enroll plugins*. Then, to use the feature **Enrolling Users to a Category:**

1) Go to the category into which you wish to enroll users. Note: You need to have category rights (manager or administrator)
2) In the Administration block, click Assign roles, which invoke the feature: **Enrolling Users to a category**. The list of possible roles you can assign users to will appear as in Figure 26 below:

Figure 26. A Moodle Screen for Assigning Roles in Categories

3) You should click on the desired role in order to start assigning users for this role inside this category. The number of user with role in the in Figure 26 above will be updated automatically after the assignments, using the feature: **Assigning teachers or students to a category**, which is a sub-feature of **Enrolling Users to a Category**.

NOTE: The roles you see here are roles which have been assigned at the category context and which you are able to assign i.e. you have to go to the module *Roles and Permissions* > *Assigning Roles* > *Context and Roles* > *Course Category Context*.

To use the Feature **Context and Roles** in the submodule **Assigning Roles**:

- In Moodle, apart from the site administrator, users do not normally have a global, site-wide role. In other words, even though you may be a teacher offline, when you are in Moodle you could have a teacher role in the course you teach in but a student role in another course where you are studying for a masters or diploma. There are a few exceptions but this is generally the case.
- Because of the way Moodle works, assigning roles is done for a particular context. A site and course are examples of two different contexts. When you create a new

role or tweak a pre-existing role, you are asked in which context(s) you want the role to be assigned as shown in Figure 27.

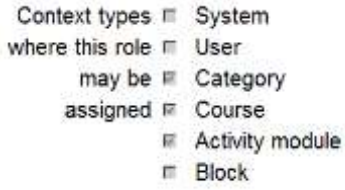

Figure 27. A Moodle Screen for Context Types where a Role may be Assigned (Moodle 2016)

In the case of this scenario it is needed to assign the role to **Course Category context**:

- Users may be enrolled in the category to save enrolling them in each individual course in that category. Used in module: **Category enrolments** as explained above.

The Problem statement for this scenario is: Suppose when the end-user uses the feature **Enrolling Users to a Category**, the list of roles is empty as shown in Figure 28 below:

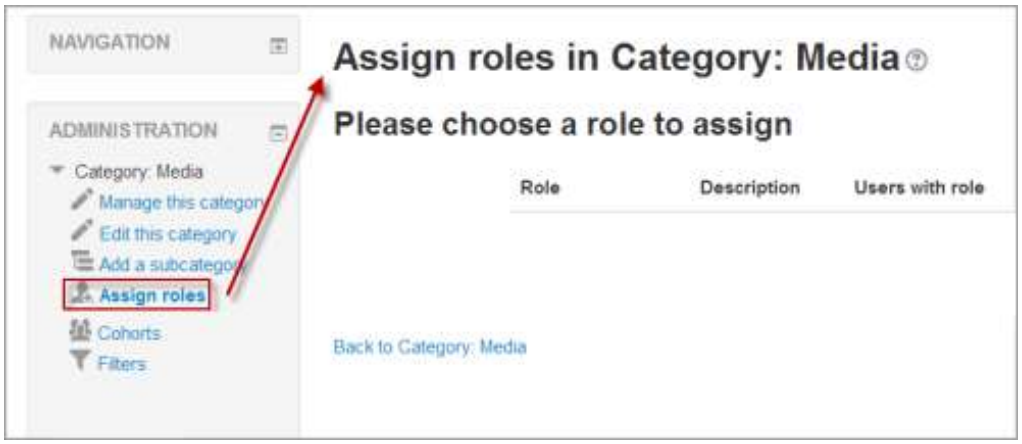

Figure 28. A Moodle Screen for an Erroneous Outcome of Assigning Roles in a Category (amended from Moodle 2016)

The end-user should use feature: *Context and Roles > Course Category Context.* However, to do so he will have to go to a different page in a different module. Write a feedback to request an enhancement, which is adding a permissions link in the Category administration block to improve the usability, and reduce the navigation steps needed to finish the task.

The suggested solution for this scenario is: In the Erroneous screenshot shown in Figure 28 that the user sent the list of roles is empty, because the user did not define the roles and allow them to be assignable in the course categories. The user is requesting to be able to

do so from the same screen and therefore, a link to the “Permissions” will be added in the administration block as shown in Figure 29 below:

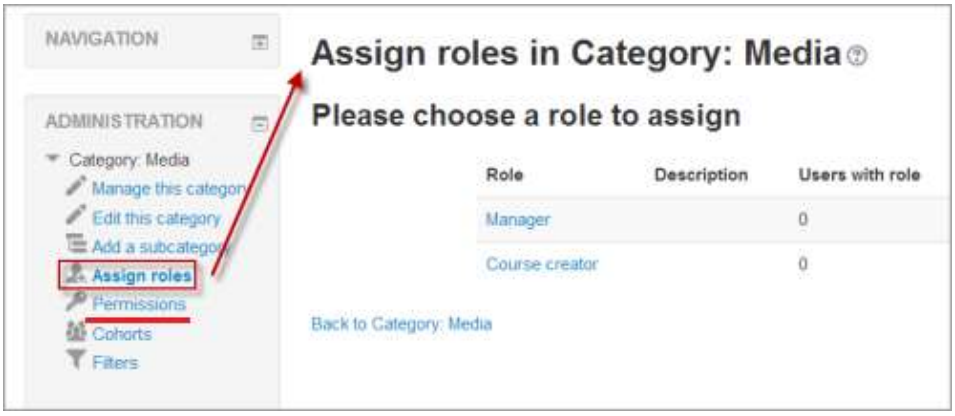

Figure 29. The Modified Moodle Screen for Assigning Roles in Categories. (Moodle 2016)

By clicking this permissions link will open to the feature “*Assigning Roles > Context and Roles*” in the **Roles and Permissions** module, which as explained in the feature description above will open the context types where this role is allowed to be assigned as shown above in Figure 27, where **Category Context** should be selected.

6.2.2.2 RE Models Utilization

For each scenario distributed in the design sessions, a business process model (Group 2006) and feature model were designed and distributed to both the users and engineers as shown in Figures 30-35. The intention from using them in the session was to help customers utilize them in their feedback and relate the problem to a certain activity in the business process model (BPM) and consequently be able to choose the related feature according to their activity choice from the BPM. Using the BPM by customers is beneficial, because it is a formal way to represent their business process yet in an easy to understand manner. They are aware of the business process through their daily work.

So, instead of searching in a large set of features and their relationships in the feature model (Kang, Kim et al. 1998), they can select the business process task they were performing and the narrowed down list of related features will occur. Arguably, this input from the end-user would help the engineers narrow down the modification scope and easily determine the impact of a change request reported by the customer.

Also, by relating feedbacks to certain activities in the business process this could help determine the source of problems in bug fixes. In order to utilize these models in the communication process, the feedback templates contained dropdown lists with the components of these models that are updated according to the users’ choice as explained in the next section.

Furthermore, since it is argued that RE models are important inputs during the analysis of issues. Therefore, it is important to design with the engineers how the requirements information can be kept up-to-date. This should be ensured by applying the modifications that resulted from new features added, enhancements in the system or even bug fixes to the system's feature model.

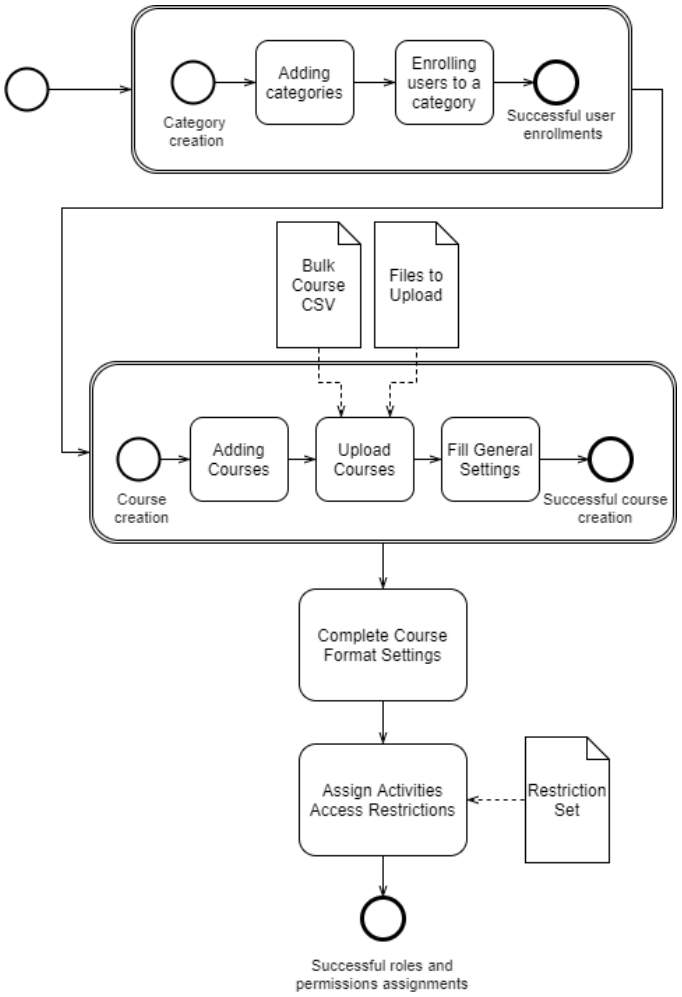

Figure 30. The Business Process Model for the scenarios used in the Design Sessions

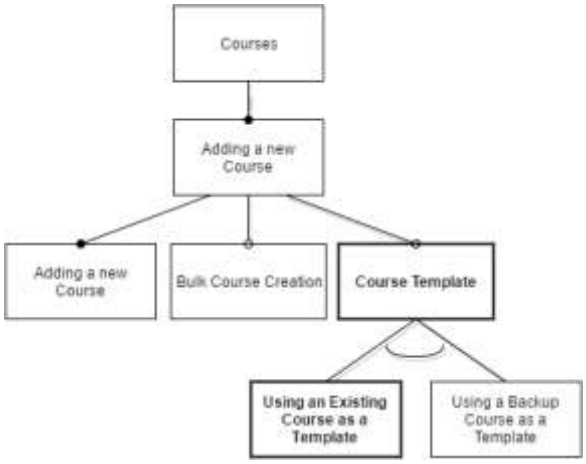

Figure 31. A Feature model for "Adding a New Course" Module

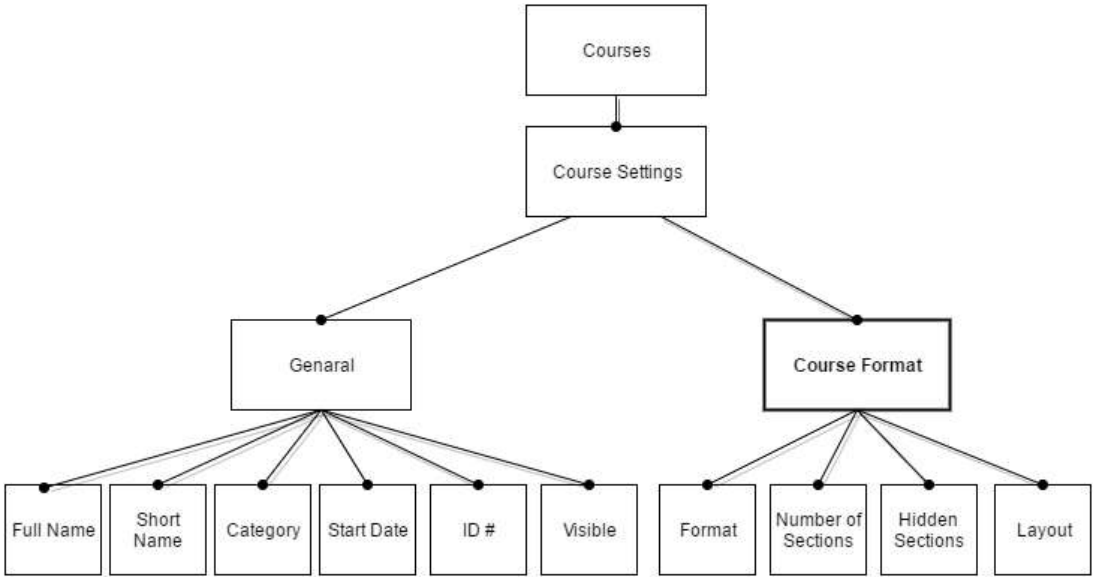

Figure 32. A Feature model for "Course Settings" Module

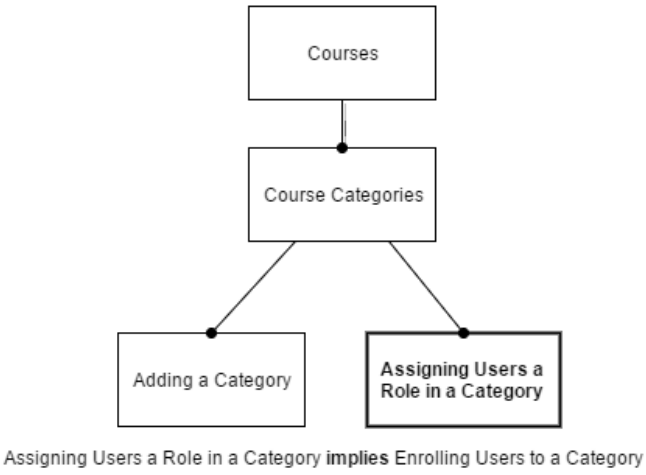

Figure 33. A Feature model for "Course Categories" Module

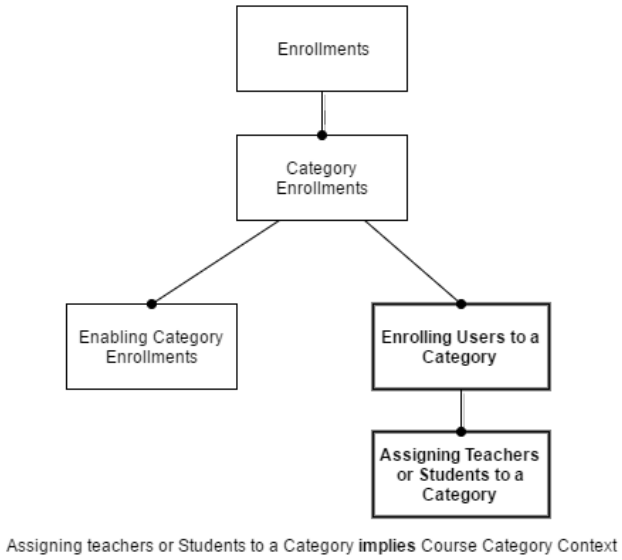

Figure 34. A Feature model for "Category Enrollments" Module

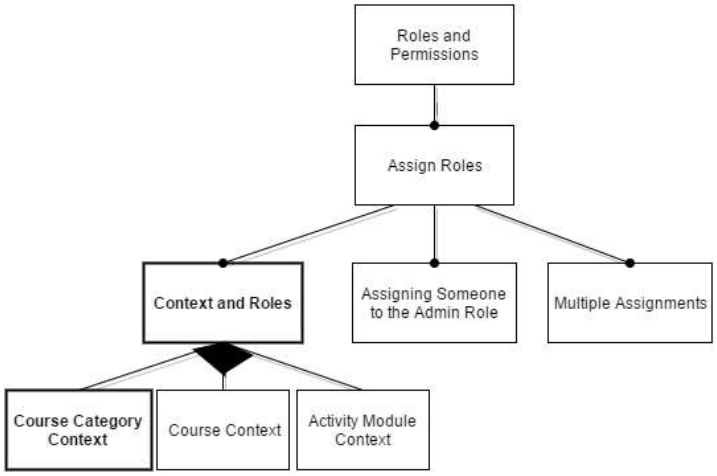

Figure 35. A Feature model for "Assign Roles" Module

6.2.2.3 Feedback Templates and Toolboxes

Mock-up techniques are ways to make effective use of the users' experience and knowledge, as well as ways of experiencing the future and they can be very useful early in the design process (Clement, McPhail et al. 2012). In the UTOPIA project, mockups became a central participatory design technique, used for example to envision technology not yet accessible (and not even heard of by the users) that could support and enhance the users work situation (Clement, McPhail et al. 2012).

In the participatory design sessions, for each feedback type, a template was designed (as shown previously in section 6.2.1.3). Each template contained the mandatory components that conform to the feedback type's rules that were extracted from the forums analysis study

and implemented as ontology. Yet one of the study purposes was to validate these rules in practice when used as a method of communication by both the end-users and engineers.

Each session of the design sessions had its own communication flow and whenever the user or engineer wanted to write a feedback, the moderator suggested the suitable template and discussed its definition and components. An example of a feedback template for the feedback type Topic Definition is shown in Figure 36. Also, for each component in the template the user was given choices for the method(s) to describe this component, which are: textual descriptions or questions using Attempto Controlled English, screenshots, links or files. Furthermore, for each feedback the participant was asked to link to RE models as shown in Figure 36 on the right side, or link to other allowed feedback types to ensure threading is kept consistent.

The participants were informed that the feedback type definition, scope, components, and the roles responsible for using the feedback type are all subject to modifications in case they need to re-design it to better fit their needs. The participants were given a toolbox to help them in case they needed to re-design the template mock-up. This toolbox contained all the levels of detail that were developed from the forums analysis study. These levels of detail consist of two categories: context and depth. These concepts were discussed with the participants in the introductory session. However, they were not expected to memorize all these concepts' definitions and usage. Therefore, toolboxes' descriptions for Depth, Context, and Attempto Controlled English were prepared, printed and distributed in the start of the session to be utilized when needed. Sample Depth Toolbox description is shown in Figure 37, while the other toolboxes are illustrated in Appendix 4 section 10.4.1. Whenever participants needed to provide recommendations, all they needed was to add components into the template body (drag and drop from the toolbox), and insert their descriptions in the text area using any of the provided methods. Any added component had a name tag above the text area that contained the component's name.

Attempto Controlled English (ACE) is a controlled natural language (Fuchs, Kaljurand et al. 2006), i.e. a subset of the Standard English with a restricted syntax and restricted semantics described by a small set of construction and interpretation rules. ACE can serve as knowledge representation, specification, and query language, and is intended for professionals who want to use formal notations and formal methods, but may not be familiar with them. Though ACE appears perfectly natural – it can be read and understood by any speaker of English – it is in fact a formal language. ACE and its related tools have been used in the fields of software specifications, text summaries, ontologies, rules, and querying.

From this perspective ACE is argued to be a good method to formalize the text written inside the feedback structures without hindering the participants' experience as it could be used as normal English that the participants are used to. Also, ACE could provide guidelines for users who could not express their problems and ideas easily. Furthermore, it has a

plugin for ontologies that could be used to validate each feedback instance entered in the ontology's knowledge base. Finally, utilizing it may well be useful when querying the ontology knowledgebase to obtain text summaries needed for feature specifications.

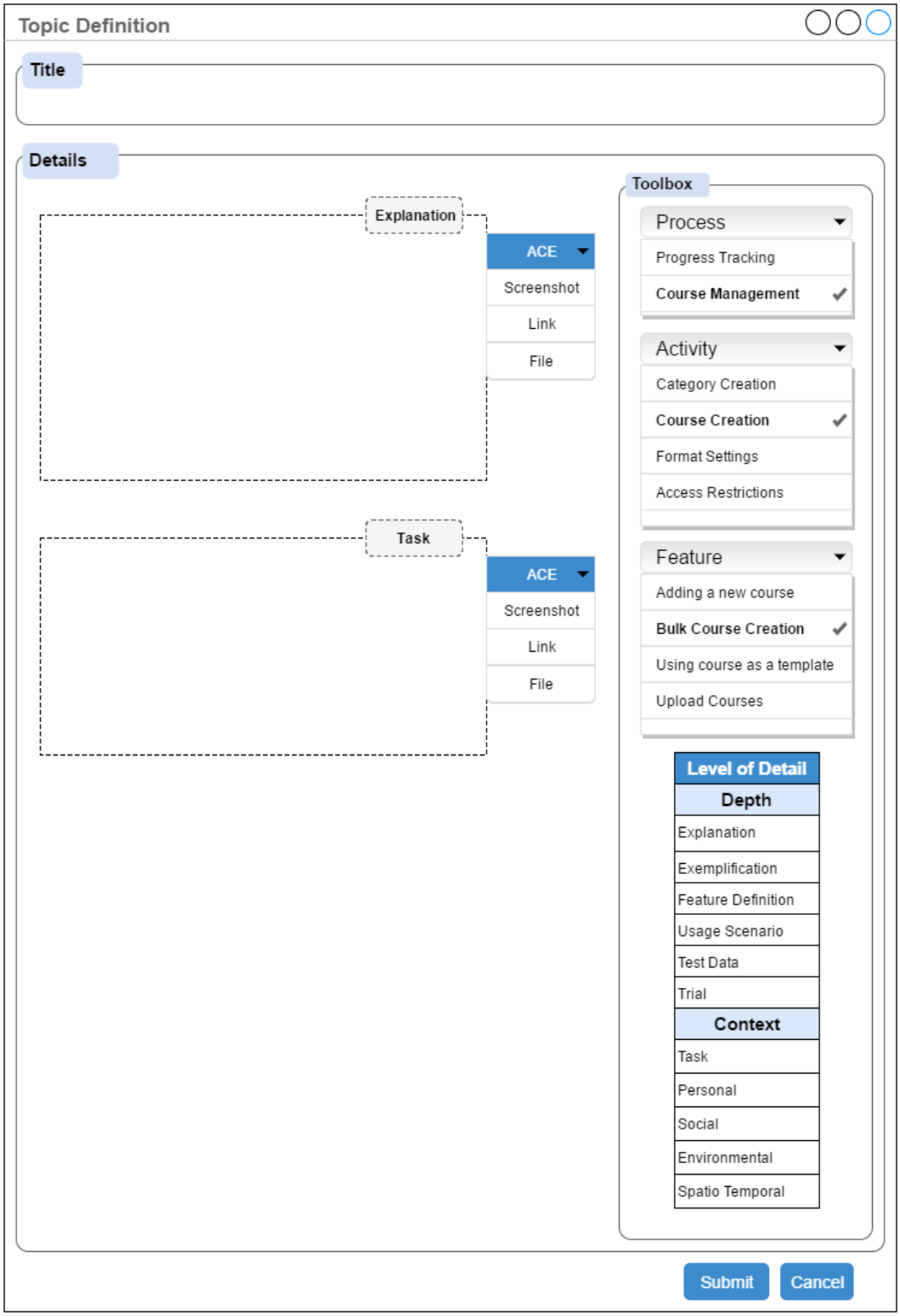

Figure 36. A Sample Feedback Template for the Feedback Type Topic Definition

To use this Depth Type in your feedback, drag and drop the box from the list into your feedback window and write your feedback.

Level of Detail
Depth
Concise
Explanation
Exemplification
Feature Definition
Usage Scenario
Test Data
Trial
Context

Concise

You can use concise descriptions to provide very short feedback with no explanations or details. For example, if you want to agree or disagree on a feedback.

"I'm having a similar problem and the above solution didn't work."

Explanation

It is always preferable to give more details in your feedback. However, it is obligatory to use explanations when explaining Problems,Solutions, or giving Elaborations to make your feedback more meaningful.

"There is a problem with the site's security certificate..."

Exemplification

Exemplification is a Depth category that is utilized when you need to provide examples within this text.

"...one with a .tmp file extension, the other has no file extension. For example, in one case the files were named 3F04D520 and 31545502.tmp."

Feature Definition

This is a depth category which the users use to define their perception of the usage of a certain feature. Engineers may use it in Solution Feedback as a way to document how to use a feature with certain types of tasks. Or with Problem Corrections, where the Engineer corrects the misunderstanding of a user by providing the correct feature definitions to features referenced in the problem statement.

"MBOX is a standard plain text file format for storing e-mail messages on hard drive."

Usage Scenario

Scenario is a Depth category which the user uses to explain text in a list. A solution can be explained in steps. Other users may list the problems they have in the problems statement.

"1. Please go to Review Tab, 2. Click the icon in right-bottom of tracking section, 3. Check and tick the box of "Pictures by Comments" to solve the problem."

Test Data

Test Data is a Depth category which the user uses to either provide sample configuration files that describe their environmental context, or to provide sample test data for the data formats that were used when the problem occurred"

OrderDate	Region	Rep	Item	Units	UnitCost	Total
1/6/2015	East	Jones	Pencil	95	1.99	189.05
1/23/2015	Central	Kivell	Binder	50	19.99	999.50

Trial

Trials is a Depth category used as a kind of extra explanation with the problem description to show the user's attempts to resolve his problem, but has failed to reach a solution. The user posts these trials of the solution and how it occurs to avoid getting similar suggestions with same trials that he already made.

"I have tried reinstalling Microsoft SQL Compact Edition 2005 and reinstalling Office 2013. I have run out of ideas and am a loss what to do next?"

Figure 37. A Toolbox for the Level of Detail: Depth with Examples.

6.2.2.4 Initial Feedback Acquisition and Communication Process

This process resembles the feedback acquisition and communication method that will be elaborated during the study. However, in order to develop this method an initial design was developed to some of the main components of this method as shown in Figure 38. The *Feedback Templates* will provide guidelines to the customers and engineers while providing their feedback, and will ensure minimum amount of mandatory information is entered and validated. These templates bound to the rules defined in ontology to govern their creation, usage, and validation. Moreover, the *Communication Workflows* represent the interaction process that the customers and software engineers undergo during the change management. The proposed workflows are aligned with the well-established software evolution process summarized in Figure 14, but with augmented tasks that integrate our feedback templates, and interactions that include customer involvement.

In this section the process in Figure 38 is going to be explained and compared to the original process in Figure 14. First in Figure 14 the software evolution process is triggered by a change request that is sent by the customer to the software engineer. Instead the new process is initialized by a formalized feedback type named: *Topic Definition*. In Topic Definitions, customers can report help requests, bugs, enhancements, or new features. Furthermore, another entrance point was added to this communication process, which is *Feedback Elaboration,* where customers can provide extra information related to a feedback already given in order to clarify some points, or add extra information that is seen necessary.

Both feedback types will go to the next phase that is problem identification, where the software engineer will try to reproduce the problem in order to identify whether it is a valid problem or not. The original process shown in Figure 14 only shows this task without catering for the different situations that may occur depending on the output of that task. In the new process in Figure 38, if it is not a valid problem, meaning that the customer reported a bug while it is a help request, or reported a bug while in fact there is lack of customer experience with the features and their usage. In this case, the software engineer corrects the customer's understanding by sending him a *Problem Correction*, and how the task should be performed and the communication ends. In case it is a valid problem, the software engineer detects whether there are missing information in the customer's feedback that need to be collected or not. If there is extra information necessary to understand the problem well, then the software engineer sends an *Investigation* with the levels of detail and questions that need to be considered. The customer is expected to answer all these questions in a timely manner using the *Investigation Elaboration*, as the whole process will be on hold, and the next phase will not be reached until the information is collected. The investigation Elaboration template requires from the customer to answer using the same types of levels of detail to ensure all the engineer's questions were answered. Otherwise, if

it is a valid problem and all information are sufficient then the software engineer moves to the next phase, which is the analysis phase.

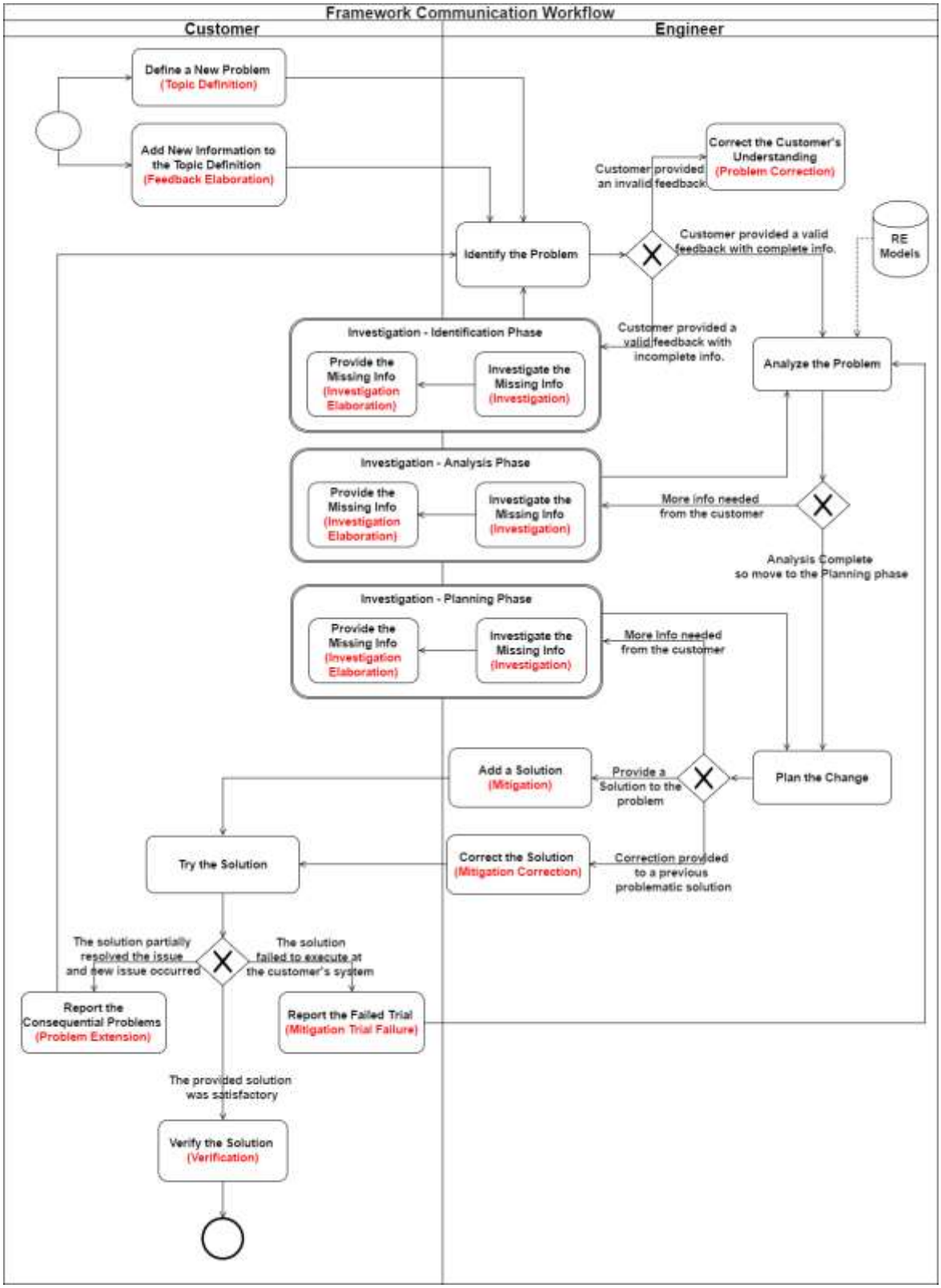

Figure 38. The Initial Feedback Acquisition and Communication Method

In the analysis phase, the software engineer analyses the problems technically in order to identify its impact on the other features or modules in the system. This will affect both the solution design and the estimates, which directly affects the customer's acceptance to the solution. In this acquisition and communication process requirements engineering models were utilized, as they would arguably help the engineers narrow down the modification scope and identify it correctly. Currently, as investigated from the interviews study that the engineers rely on their own experience and code reviews, which is a very exhaustive method and depends completely on the engineers' knowledge with the code. However, if the engineer working in the project was no more involved, or the customer requested a modification in a less frequent module, this could lead to errors in impact analysis. In case the information gathered from the previous phase, and the RE models employed provided enough information for the engineer in this phase then the flow moves on to the planning phase. Otherwise, he will investigate the information needed, and the customer will be required to provide investigation elaborations for all the questions asked.

In the planning phase, the project manager produces a workable plan including all the information gathered by the software and requirements engineers in order to produce a feasible solution within a correct scope and reasonable estimate. If any extra information or negotiations are needed to be made with the customers for example on the priorities and release plans updates, they are made in the planning phase through investigations asked by the project managers and investigation elaborations that provide answers by the customers.

The original process in Figure 14 shown that the Solution is then implemented by the developers and delivered to the customer, whom in turn tries the solution and reviews it. In the new process in Figure 38 shows that the engineer should provide *Mitigation* that describes the solution and its usage scenario, and explains the updates. In case the customer tried the solution and it completely resolved the reported issue, then the customer should provide *Verification*, and the communication process ends. Otherwise, if the customer could not try the solution, then he reports a *Mitigation Trial Failure*, which will return to the software engineer for analysis, planning and implementation update, and then a *Mitigation Correction* should be provided back to the customer with the updated solution. Else, if the customer tried the solution and it partially resolved the problem, but new issues evolved, then this situation should be reported as a *Problem Extension*, where the new aroused issues will be directed to the identification phase in the beginning of the process to be treated as a new problem.

In the introductory session the initial draft of the new process was introduced and explained to the participants. Also, it was contrasted to the original software evolution process as explained in this section, to criticize the gaps and show the strengths of the new design. Also, when the sample case study was described in the end of the introductory session, the undertaken flow was indicated.

6.2.2.5 Feature Specification Template

In this section, the last outcome in the design session is explained, which was related to the feature specification documentation. After introducing the problems of gathering requirements and current challenges in the introductory session, the purpose in the design session was to design a new systematic method for gathering new requirements or requirements' updates using the new classification of feedback structures. These structures are stored in a knowledge base that can be queried to gain any type of specific level of detail from the feedback thread that is needed to update the feature specification document. The session's aim was to: 1) determine the suitable structure for the feature specification document; 2) decide on the feedback types that provide the needed information to update the feature specification document; 3) develop a process for acquiring information and updating the documentation.

Figure 15 provides a sample feature specification document that was the starting template during the design session (Robbins 2004). Each participant received a copy of the template and they started to brainstorm all the issues and possible designs for the template and process. Also, Figure 39 shows a sample feedback thread that was provided to the participants during the session to help them imagine how feedback types could be related to form a thread. Also, during the design session they have gone through all the feedback templates and understand its components. Thus, by the end of the session when they reached this point they have a holistic view of the types of information stored in the knowledge base and can provide useful insights on how this information can be utilized to fulfil such aim and justify their designs.

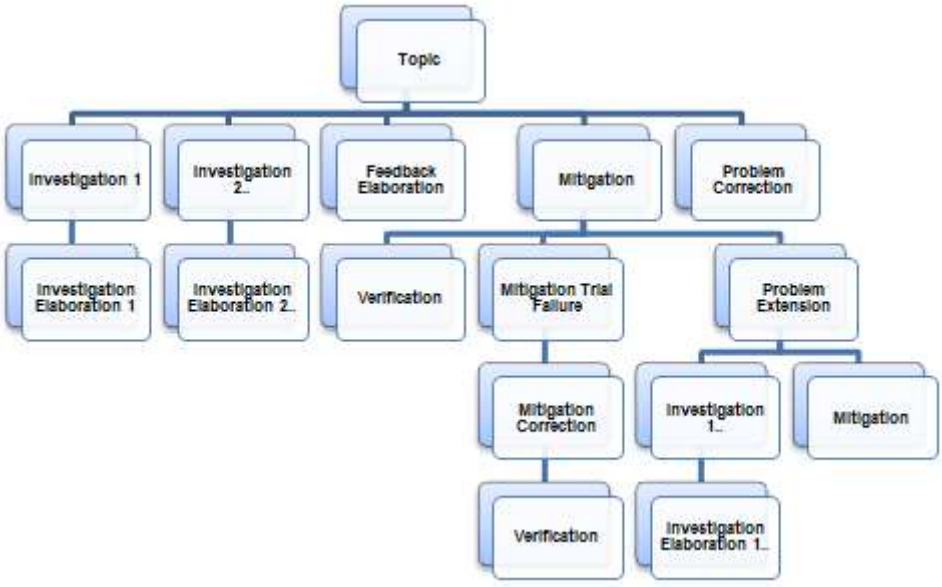

Figure 39. An Example for an Interrelated Feedback Thread

6.2.2.6 Questions asked

Below is the list of questions that were asked during the design sessions. The questions are categorized according to the topics that were under investigation/ evaluation in the study.

Templates Question(s):

1) Do you find the provided feedback template components sufficient for providing the needed information? If yes please justify why, and if no please propose your own recommendations of other components and provide justifications.

Attempto Controlled English (ACE) Question(s):

2) Do you think using the controlled English hinders your experience in providing feedback easily? Or do you find it more helpful in structuring your text? Justify your opinion.
3) What are the components in the ACE toolbox you find more important or useful than others? What are the components you find unnecessary or hard to use?

RE Models Utilization Question(s):

4) Do you find that the models employed during the process were useful? If yes, please explain how they improved the task execution. Also, propose your recommendations of how can the models be kept up-to-date.

Feedback Acquisition and Communication Workflow Question(s):

5) Do you think that the proposed communication process overly expands the time in which the tasks/ issues are handled? Or do you think that ensure better communication to save time and effort that were going to be spent afterwards? Elaborate on the advantages and or possible drawbacks you find in the process.
6) As an end-user, do you think that this process imposes extra obligations in your involvement? Elaborate on the gained advantages of this involvement and/or possible difficulties or issues.

Requirements' Specification Template Question(s):

7) In each task in the evolution process phases: what is the information needed to update the feature specification document with?
8) For each phase: Which feedback templates and/or RE models will you use to gather the needed information to update the feature specification document?
9) What are the sections that need to be added or updated to represent the gathered information in the feature specification document?

Notes and Summary Question(s):

10) Do you have any comments, suggestions or advice about our work that you would like to share?

6.3 Study Results

In this section the results reached from the participatory design study will be explained. The results are divided into 3 main categories: 1) updates in the feedback acquisition and communication process, 2) updates in the feedback types' classification and/or definitions, and 3) design of feature specification extraction process. Also, there are results relating to the participants' opinions in: 1) end-user involvement in the process, 2) feedback components restrictions and ACE utilization, 3) RE models utilizations.

6.3.1 Feedback Acquisition and Communication Process Updates

The feedback acquisition and communication process was updated according to the workflow that the end-users and software engineers went through during the design process. They were led through that flow according to their own feedback inputs and communication during the session. For example, some users started the communication thread by reporting the problem in the form of a help request (i.e. they want to perform a certain task and don't know how). Others reported it as a bug that needs to be fixed, as they do not understand the output that occurred when performing the task, or they were expecting a different output to occur. However, in these cases and after communication with the engineers they realized that the feature already exists in the software, but they want it modified to fit in the task they were doing. Other users directly reported it as a change request as they know a workaround for performing the task, but they want it re-implemented in a different manner.

A sample real thread of feedback communication is illustrated using figures in Appendix 4, Section 10.4.2. Also, the updated acquisition and communication method's workflow is shown in the same section, along with a feature specification document with the participant's suggested updates.

6.3.1.1 Sample Design Sessions Threads

In this section examples from the PD design session threads are illustrated to show how the communication between end-users and engineers took place. This communication took place through the designed feedback templates for each feedback type as shown in Figure 36 and explained in section 6.2.2.3. As a result the feedback acquisition and communication process evolved and the essential amendments were made. Before the amendments are

explained in the next section 6.3.1.2, the key sessions that led to these adjustments are shown here.

As mentioned before two fictional scenarios were used in the PD design session to immerse the participants in a problematic situation that could lead to enhancements in the current system software. In this section, three distinctive design session threads are detailed, where thread 1 and 2 used scenario 1, while thread 3 used scenario 2. The other two sessions were not presented, because they shared common decisions that were made in the other sessions.

6.3.1.1.1 Communication Thread Sample 1

1) The end-user provided a **Topic Definition** explaining the enhancement she wants to make to the system. Relating the feedback template the business process, and business activity and the feature that needs to be enhanced.
2) The end-user then provided extra information that she did not provide in the Topic Definition. So she provided a **Feedback Elaboration** that related to the Topic already defined, specified the level of detail that needs to be provided, and wrote the feedback.
3) The engineer identified the feedback as valid problem, but still she needed to confirm the new requirement with the end-user. The **Investigation** template was used where a textual explanation of the perceived information was provided and then a concise question was given asking the end-user to confirm. Also, it related to the Topic.
4) The end-user provided an **Investigation Elaboration** where a concise level of detail was used (same as the level of detail used in the Investigation template). The end-user also related to the Topic and Investigation.
5) The end-user then provided a **Feedback Elaboration** to add extra information that was triggered from the communication. In the template a feature definition level of detail was chosen, and a textual description was added. Also, the template was related to the Topic.

In the initial draft of the new method shown in Figure 38 it was suggested that in each of the main phases a loop of investigations could take place in case the software engineer needs to collect further information needed to identify, analyze, or plan the issue. In order to ensure information are gathered in an accurate manner, the engineer can send questions in an *Investigation* on any level of detail he wants to ask about, while it is restricted that the feedback type *Investigation Elaboration* provided by the end-users must contain only answers to the same type of level of detail.

However, during actual communication as shown in this communication thread 1, it happened that during the investigation loop end-users needed to communicate extra

information about the requirements that they did not put in the main *Topic Definition*, or the Investigations asked by the engineers raised further concerns/ constraints that they felt they need to communicate. This was not catered for in the original process and therefore they suggested that *"there should be an extra step for end-users to provide the extra information they want at any time".* The participants suggested *"using the Feedback Elaboration template as it is most generic feedback type for that purpose"*. Also, the engineer participants suggested that since these extra information could affect the feature definition, scope, and/or add extra details, therefore *"they should be reviewed by the software engineer and directed to the identification phase"* to determine whether it fits or contradicts the existing definition, and decide whether it will affect the solution design, and whether it will be handled in the current situation or planned for in next releases.

The modification was settled on adding a decision whether the end-user would like to add extra information after providing each investigation elaboration at any of the main phases of identification, analysis, or planning (i.e. before the implementation of the actual solution begins). If the end-user decision was yes then they will use the Feedback Elaboration to add information and it will be directed and studied in the identification phase. If not then the decision will be directed back to the same process phase it was in before the investigation started. This is shown in the final process documentation in Figure 40.

Also, the session moderator asked the participants whether it possible that the end-users could add extra information during the implementation and the engineers answered *"yes they could, but anyway it will be catered for in another version, as it will be treated as a new issue but related to an existing problem, and therefore it should enter the maintenance cycle from the beginning".* This is shown in step 11 in this sample communication thread.

6) The engineer then started to investigate the newly added information. An **Investigation** template was provided carrying several questions of the same level of detail that is feature definition. Also, the engineer related that to the Topic.
7) The end-user provided an **Investigation Elaboration** template to answer the questions that the engineer asked using same type of level of detail (i.e. feature definition), related the feedback template to both the Topic and the Investigation it answers.
8) The engineer used the RE models to determine the problem scope, analyze the problem, and think of the possible solution to the enhancement and what possible modifications it will make to RE models. Initial plans and solution design were made. As mentioned in section 6.2.2.1 that a suggested solution was provided to the engineer to be used in the Solution templates. However, the engineer suggested that a **Proposal** phase and template should be added before the Solution.

The engineer agreed that *"a Proposal to the solution has to be made first before we proceed to the actual implementation especially in case of enhancements or new features in order to discuss it with the customer first".* However, in case of bug fixes a proposal might be not necessary *"for example if it a problem in the UI where the indentation is reported then obviously the fix will resolve that matter without a need to propose first"*.

Therefore, a new activity was added to the final version of the new method, shown in Figure 40, named **Add a Proposal** to provide such output and it employs a new feedback type named **Proposal**, which is a subtype of the Mitigation. This activity can be performed optionally when needed after the planning phase, where the engineers provide an initial suggestion of the solution design and plan. If this proposal was acceptable then a **Confirmation** should be provided. The engineer used the same types of levels of detail of the Solution template and related it to the Topic feedback.

9) The end-user provided a **Confirmation** to positively verify the suggested proposal that was detailed by a concise description and personal context. It was also related to the Proposal.
10) The engineer then provided a **Solution** template that mainly carried same details of the proposal since it was confirmed by the customer. And the engineer suggested that it should be related to the Proposal feedback type.

The engineer suggested that *"since the Solution feedback template contains a usage scenario level of detail to explain the solution usage in steps; these steps could contain information for linking the enhancement to other existing features in the feature model"*.

The solution template is provided to end-users as part of the documentation used when they are trying the solution, and therefore, this could help them understand the feature usage better. Same as in the Moodle documentation shown in section 6.2.2.1, where some features relate to others in a different module so a link was provided that directed the user accordingly.

11) The end-user then decided to provide extra information that adds new modifications to the recently implemented enhancement. So she provided a **Feedback Elaboration** detailed by textual explanation, social context and Spatio-temporal context.
12) The engineer decided that this modification will be handled as a new enhancement and will go through the same process.

6.3.1.1.2 Communication Thread Sample 2

1) The end-user wrote a **Topic Definition** explaining his problem, and questioning whether this is an existing feature that fulfills his request. This template was related

to the BPM process and activity and accordingly to the feature the end-user was using.

2) The engineer provided an **Investigation** asking him several questions with the aim to clarify the feature definition of the requested enhancement and related it to the Topic.
3) The end-user provided an **Investigation Elaboration** where he answered the questions using the same level of detail feature definition, and related the template to the Topic and the Investigation templates.
4) At this point the engineer has understood the problem and the Identification phase ended. Followed by an exploration of the RE models provided to analyse the modification scope and suggest a possible solution. In this session the engineer also suggested that a solution **Proposal** should be presented to the end-user to demonstrate the solution usage scenario and screenshots to exemplify how the solution would look like.

As suggested by participants in the previous thread, if the end-user accepts the Proposal then a **Confirmation** should be provided. This was suggested in this session too. This **Confirmation** is suggested to *"go back to the planning phase to complete the plan with finalized estimates and resource planning then the actual implementation can proceed."*

Moreover, in this session the engineer discussed the other possible outputs from the proposal phase. If the proposal was not acceptable, the engineer suggested that a new feedback type could be designed named **Negative Verification**, which end-users can use to negate **Proposals** only and provide textual justifications explaining their reasons for the rejection. Moreover, the participants explained that after a *Negative Verification "it could happen that the customer totally rejects the proposal for example for financial reasons then the process ends"*. Else, *"the customer may choose to propose an alternative solution or make modifications"*, and in this case the process directs them to the Feedback Elaboration template (as this is the most generic template where end-users can specify any level of detail the provide their feedback) in the start of the process where they add the information they want to an existing problem and the engineer starts handling it through identification, analysis and planning.

Finally, a minor change was made to the process to generalize the results. Since, the Feedback type **Confirmation** will be used to represent positive verification; therefore, the final method version in Figure 40 was also updated. The verification after a Problem Correction and after the Solution is also made through a Confirmation feedback type, followed by an end-state to show that this is a final acceptance by the end-user.

5) In this step and after the end-user received the Proposal from the engineer, he provided a **Negative Verification**, where he rejected the proposal and used

personal context description and justified that he wants to make some modifications in the usage scenario provided in the Proposal.

6) The end-user then provided a **Feedback Elaboration** with a usage scenario level of detail explaining the modified steps he wants the feature to work according. The end-user related the feedback to the Negative Verification.
7) The engineer then identified, and analyzed the modification and found it valid and feasible. Then she provided a **Solution** with the modifications applied to the usage scenario and a screen shot exemplifying the solution. The engineer also related this feedback template to the Topic Definition feedback.
8) Finally, the end-user made a **Confirmation** on the solution to verify that he made his final acceptance. He detailed his confirmation by textual concise description.

6.3.1.1.3 Communication Thread Sample 3

1) The end-user provided a **Topic Definition**, however he did not explain his problem as an enhancement, but instead he wrote his feedback as a bug report. The end-user user the explanation level of detail and used a screenshot to exemplify the erroneous output.
2) The engineer provided a **Problem Correction** feedback type in order to clarify that this is not a bug in the system as the feature exists. The engineer provided a textual feature definition for the feature in the system that could be used to fulfil the task, and also he linked it to the feature specification document for that feature (this was provided in the scenario 2 description as shown in section 6.2.2.1). Also, he detailed his feedback with textual explanation of how the feature should be used and provided a screenshot for the part of the system the user should be configuring to manage the task correctly. Then he linked the feedback to the provided Topic.

The engineer then recommended that he *"would like to use more than one method to describe the feedback type"*.

3) When the engineer clarified that what he is requesting already exists, the end-user replied that he wants an update in the feature and he wants it to be accessible on the screen he was performing the task on, which means he was requesting and change in the workflow's behaviour and also, a change in the feature's scope.

In the feedback types' definitions developed in this research (Sherief, Abdelmoez et al. 2015), there exists a feedback type named **Addition** that could be used in such case, because by definition this feedback was used to add a problem to a problem. However, from the forums analysis that was conducted in this research the original definition of Addition was used when end-users added problems not related to the main problem on which the discussion is held. That is why; this feedback type was not employed in the initial draft for

the process, because such flow was thought to appear in online forums where communication is less organized and loose, but not in business context. However, to use this feedback type in the new communication process, a more refined definition was needed that was: a feedback that the end-user can use to add a new enhancement after a Problem Correction was provided, meaning that *"the feature definition and scope is no longer suitable for the end-user and needs modification"* as mentioned by the end-user. Also, both the end-users and engineers in all the PD design sessions agreed that *"even if the Problem Correction was a satisfactory answer to the end-user, a* Confirmation *should be provided to confirm the end of thread"* as mentioned by the engineer. Finally, this new change reported in the Addition feedback type, would be treated as a new problem and the flow moves on to the identification phase where any needed investigations are made and so on.

4) After the Identification phase the, the engineer started to analyze the situation by examining the RE models to determine the modification scope and how will the proposed solution affect the RE model.
5) After that the engineer was ready to provide the solution, however as in the previous communication threads he suggested that a **Proposal** phase and feedback type should be added before the solution to discuss the suggested solution with the customer and have his feedback. The engineer provided a Proposal feedback type detailed by textual usage scenario and exemplification described by screenshot containing the suggested UI for the solution. The proposal was related to the Addition feedback.

The planning phase was overlooked during the design session as this is supposed to contain management tasks (time planning, cost planning, resource allocations…), and since this is fictional scenario so these details are ignored. However, in all the PD design sessions the engineers agreed that *"initial planning should be made before the proposal phase, but are finalized after a confirmation is received from the end-user, because there is no need to spend effort on making a complete plan, while the proposal can still be rejected or modified"* as mentioned by the engineer.

6) The end-user then provided a **Confirmation** detailed by concise textual description, and he related the confirmation template to the proposal.
7) The engineer then provided a **Solution** template that was detailed by usage scenario that was detailed by text and link method and an exemplification using screenshot to the final solution UI was also provided same as in the proposal template.

The engineer was engrossed by the idea that the feedback acquisition and communication process specifies that RE models should be used as an input in the analysis phase to provide accurate inputs during the impact analysis and scope definition. Hence, he raised a

concern about how to keep the feature model up-to-date specially that an enhancement was made, so that these models can lead to accurate results when used in future system changes. Accordingly, the engineer started providing ideas to cater for designing such task.

First he discussed that *"Proposal and Solution templates should be used to document how the feature model will be updated"*. Furthermore, the engineer suggested that *"the modifications should be specified in the usage scenario in the Mitigation templates as part of the solution steps that reference new features.* He also clarified that *"If there were no changes in the Proposal usage scenario then it could be used as it is in the Solution template. However, if an update happened according to modifications communicated by the end-user then a new usage scenario will be specified in the Solution template and the latest version will be used to modify the feature model"*.

The engineer also emphasized on the importance of providing information about the feature models updates during the Proposal task. This is because the Proposal template is used during the negotiation with end-users, and sometimes important feature model updates need to be discussed with customers too. *"For example, in a scenario where the end-user reported that he needs a feature for a new type of report that includes specific attributes. Then the engineer proposed a solution design where a more generalized feature will be implemented, by which the end-user can define the attributes that he needs to report instead of making several static types of reports. This will lead to the addition of a new parent to a list of features. Also, these sub-features will require new relationships or cross-tree constraints in the feature model."*

Moreover, the engineer also discussed that *"the existing feature model notation (i.e. relationships and cross-tree constraints) can be used in the usage scenario"* to describe the links in the feature model. This would help make the modification of the feature model *"more accurate and systematic during the actual updates"*. As a result, the Mitigation template will contain information about how the feature model will be updated according to the modification(s) that will be/ was implemented in the system. This can result in *"adding new feature(s) and relating them to existing feature(s), or relating two existing features together, or relating two new features together where one of them is related to an existing feature"* as specified by the engineer.

8) The end-user then provided a **Confirmation** detailed by concise textual description, and he related the confirmation template to the Solution.

6.3.1.2 Integration of the Suggested Enhancements

In this section a completed view of the novel feedback acquisition, communication and requirements update method is provided in Figure 40. The key modifications that were made to modify and enhance the process are also summarized in this section.

First, the process was modified to improve the Problem Correction workflow which occurred as one of the paths after the Identification phase. Two modifications were made which are adding a Confirmation after the Problem Correction and adding an Addition feedback Types in case the end-user wants to add an enhancement.

Second, the Investigation tasks were always directed to the Investigation Elaborations which are strictly answers to the questions asked by the engineers that is then re-directed to same phase from which it originated. This was enhanced by adding a possibility to add extra information to the requirement (problem) at any time during the communication, which would then be redirected back to the Identification phase so that the engineer could study whether it is a valid addition or not.

Third a new feedback type named Proposal was suggested by all engineers in all the design sessions. This was demanded so that the communication process conveys the actual situations that happen in real situations, which is discussing the proposed solution with an initial plan with the customers before actual implementation takes place. Moreover, this was further enhanced by engineers in a particular design session (as shown in section 6.3.1.1.2) to add different outputs to the proposal task. The task could results in: a) end-users accepting the proposal thus a Confirmation task was added; b) end-users totally rejecting the proposal thus a Negative Verification feedback type and task were added; c) end-users rejecting the proposal with modifications thus a Negative Verification would be directed to a Feedback Elaboration task to add the needed information.

Fourth, an enhancement to the Mitigation feedback templates was suggested so that engineers are obligated to specify the feature model updates during the Mitigation task (i.e. Proposal and Solution). The engineer suggested the scenario should be detailed more and divided into multiple steps and that the feature model notation (i.e. relationships and cross-tree constraints) should be used in those steps to specify how features and/or links could be added to the existing feature model.

Since the engineer suggested modifications in the feedback template, and how it could be enriched to carry important information that could support the engineers in later stages when the models and documentation are actually being updated. Therefore, it was considered necessary to further update the feedback acquisition and communication process to enforce the preparation of the feature model updates, and also to inform them how to perform this task.

From the analysis results and after putting all the engineer's ideas into consideration the process was modified by transforming the "Add a Proposal" and "Add a Solution" tasks into sub processes, where a sub process was designed to guide the engineers through the steps needed to complete that task of documenting the updates of the feature model. Similarly, it was found that the "Correct the Solution" task should also be transformed to a sub process.

This task originally, was designed so that the engineers could provide a Mitigation Correction to a Solution previously provided to the end-users, but did not work properly (Mitigation Trial Failure), or another problem evolved after applying it (Problem Extension). The Mitigation Correction feedback type is detailed by a usage scenario where the corrected steps of the Solution are provided. These steps may contain modifications to the documented feature models updates. Thus, the Proposal, Solution, and Mitigation Correction tasks were all transformed to sub processes.

The sub process in Figure 41 was designed to work on the feedback types' details in order to help engineers complete their details, while putting into consideration the feature models modifications. First, the engineer starts by the main level of detail, which is the usage scenario. So in first task in the process the engineer is asked to **"Add a usage scenario step"**. This step is then evaluated, if it needs to include feature model linking then the engineer should **"Specify the Link type"**. In the task of specifying the link type, the engineer should choose from the feature model notation the suitable relationship type (Mandatory, Optional, Or, Xor), and/or cross-tree constraints (requires, excludes) that he wants to use. Next a decision should be made whether he wants to **"Link to an existing implemented feature"** or **"Link to a new proposed feature"**.

After specifying the link type and the feature type it will be linked to, two possible paths could be taken, either to go back to the "Add usage Scenario step" again to complete the usage scenario description, or go to another task **"Complete the Feedback Description"**, then the process ends. The task "Complete the feedback description" depends on which type of feedback the engineer was providing. If the engineer was providing a Proposal or a Solution then after providing the usage scenario he should provide an exemplification, for example a screen shot of how the solution would look like. If the engineer was providing a Mitigation Correction then after providing the usage scenario he should provide an explanation of why the modifications were made and/or how they were made should be given.

If the evaluation after the "Add a usage scenario step" task did not lead to the need to make linking to the feature model, then two other possible paths could be made. Either to go to the same task again which is adding another step in the usage scenario, or go to the "Complete the feedback description" task and end the process.

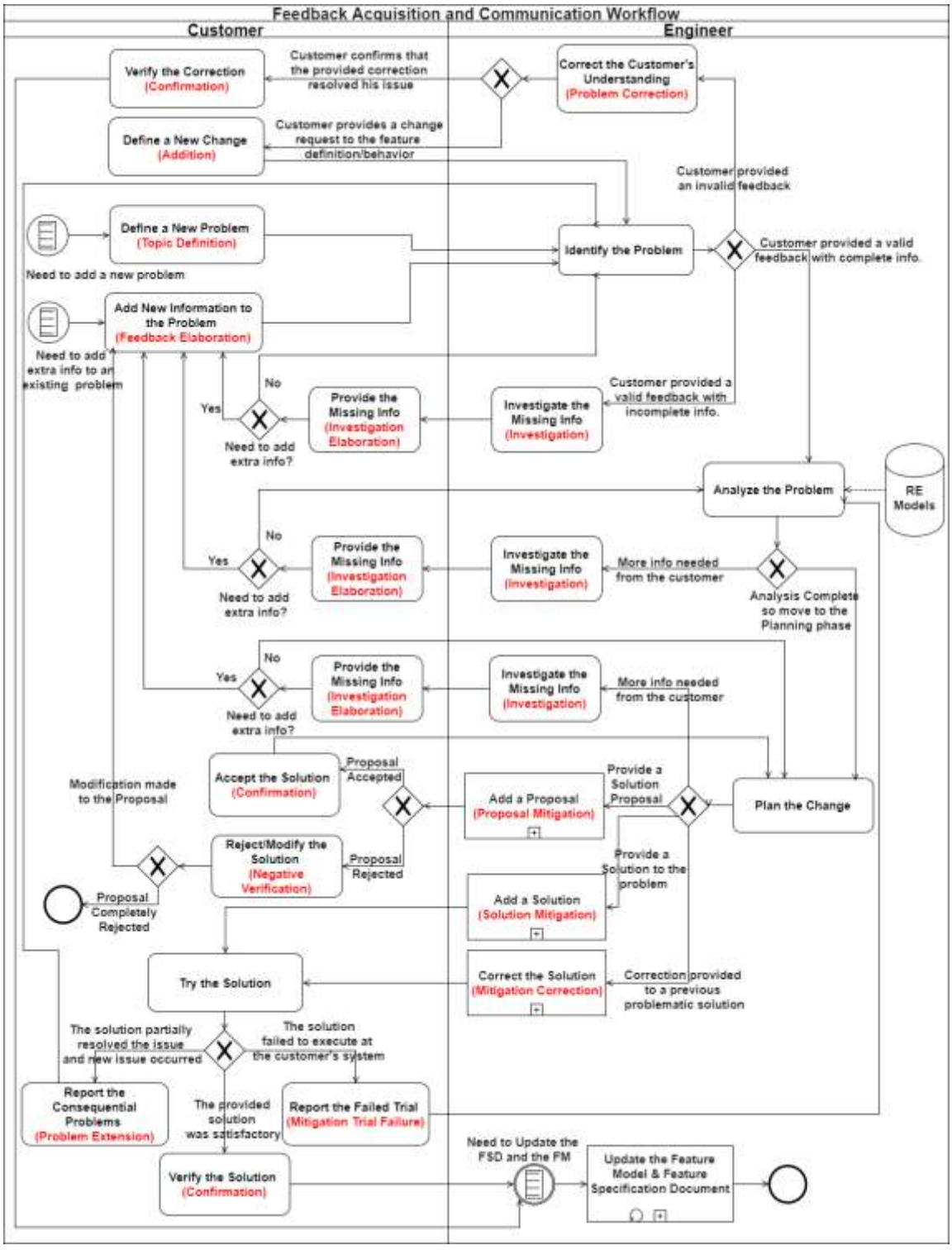

Figure 40. The Final Feedback Acquisition, Communication, and Requirements Updating Method

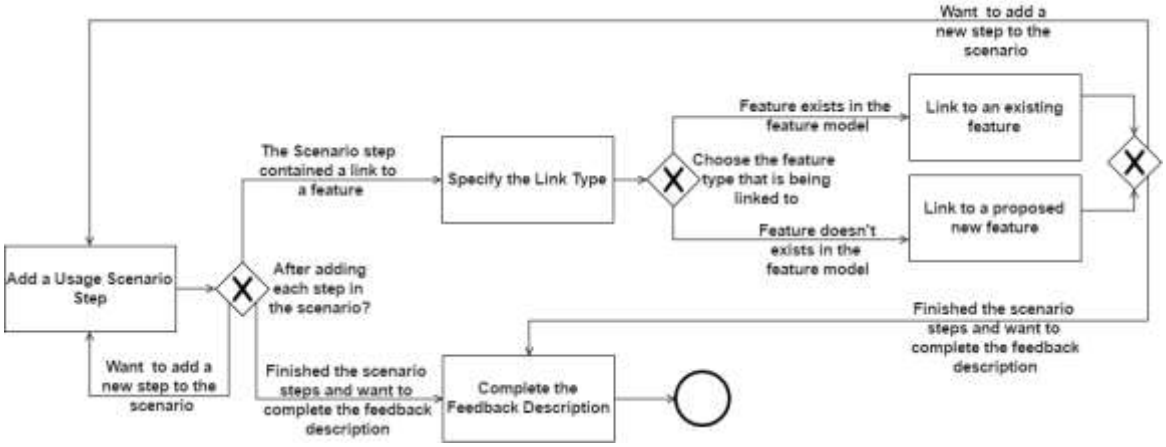

Figure 41. The Proposal, Solution, and Mitigation Correction Internal Sub Tasks that Link their Scenario Steps to Features.

6.3.1.3 Results Concerning Participants' Involvement in the Process

In this section the modifications made to the feedback acquisition and communication process were explained and justified from participants designs. Still, other questions were asked to the participants concerning their involvement in the process as seen in section 6.2.2.6 (questions 5 and 6). The engineer participants agreed that from the advantages of this process is that it *"aligns with the normal software evolution process they already perform"*. This makes it easy to understand and use, because the main phases and tasks are already there and in the same order. But, it *"adds definition and guidance to the communication tasks that were performed in an-hoc manner"*, and concerning the feedback acquisition *"regulations are enforced on both end-users and engineers through feedback types, which ensures useful and meaningful information is being collected"*.

In all PD design sessions, the participants agreed that even if this is a longer process but *"still it was performed anyway, but under stress when communication problems occur"*. When issue handling is made when communication problems accumulate, "*makes it harder to handle and documentation becomes a burden, as all what we concentrate on is to resolve issues quickly and not through a methodical manner"* as mentioned by engineers. This causes requirements information to be lost, poorer documentations and less customer acceptance. Thus, both types of participants confirmed that *"it would be better to have a more organized process where each role knows his responsibilities and scope for involvement"*. From the customers' side, this makes them *"feel safer to be more involved and know how things are handled"*. Also, the engineers confirmed that this would definitely *"lead to handling of issues in an efficient and timely manner"*. Also, the idea of communicating through feedback will *"provide better justifications to them especially in case of disputes with customers and project closures*, *where explanations need to be provided of how and why certain things were done".*

Finally, the employment of RE models during the evolution process was examined. This part was tested in two different ways. First, the feedback templates provided to the participants contained section for linking the feedback to a certain Process and Activity in the business process and accordingly the features were generated to choose from them as shown in Figure 36. This helped the customers locate problems in a more precise manner, which was argued to help engineers in tasks such as impact analysis of issues. Second, in the PD design sessions, engineers had a pre-prepared sample feature model and business process model for the Moodle that models the business activities and relationships between them and with other features in the system as shown in Figures 30-35. Engineers found the models utilization is *"useful in narrowing down the modification scope"*, as it helps locate the problematic areas in the activities carried out in the business, which is easy for the end-user to specify, and therefore *"it clarifies accurately which features may be affected by the modification".* Helping engineers in narrowing down the scope *"helps in validating the issues reported by end-users and reduces the effort spent being misled due to lack of sufficient descriptions and customer experience issues".* Furthermore, the engineers in PD design session 2 and 3 suggested that this a promising step that could help provide a base for further analysis that may be done on the links between feedbacks and RE models to obtain useful reports for example on *"which are more problematic features",* or *"features where more costs were spent on enhancements changes",* and more importantly *"maintain the requirements and feature specifications up-to-date and linked to the latest software updates".*

6.3.2 Feedback Types Updates

This section explains the updates that occurred during the PD design sessions regarding the feedback types' templates and the utilization of ACE (Fuchs, Kaljurand et al. 2006) to provide structured feedback. As explained in section 6.2.2.3, a feedback template was designed for each type of feedback. These templates conform to the rules that were defined earlier in the research (Sherief, Abdelmoez et al. 2015). However, the purpose was to validate them in practice during the design sessions. Moreover, to help participants employ the ACE structures a toolbox was provided and explained with examples.

Since the designed mock-ups were used in the sessions were paper-based and not computerized, it was suggested that participants should use small stickers to tag each sentence with the sentence type they want to write and provide the input. However, whenever the participants needed to structure the sentences using ACE they *"just assume that the component was dragged and dropped in the text area"*, and the appropriate sentence was written. This is the way they *"prefer if this was implemented in reality"*, as they clarified during the design sessions, especially that this *"labelling could be hard-coded in the program"* as suggested by the engineers participants. *"This would be easier when providing the feedback"* as agreed by both end-user and engineer participants.

For the practice of using ACE, participants had different opinions. In the design session 1 both the engineer and end-user were not keen on the idea of *"being obliged to use a certain way in writing their textual descriptions"*. They said that *"this might block them from fluently explaining their feedback especially that they always have in mind an idea of what they want to say"*. However, they pointed out that *"this depends on the type of personality for the person writing the feedback especially the end-user, because end-users trigger the communication, and not all end-users are very expressive to their problems'*. So if the end-user had an expressive personality type, his experience should not be hindered while providing such essential input. While, if the user was not expressive enough, then an ACE tool could have great benefit to him. Putting into consideration the benefits of having structured feedback, the engineer in this session agreed that having ACE is of good value especially if further analysis was to be done. Therefore, it was suggested to be put as a validation step meaning that *"after the feedback is written it could be validated against the ACE construction rules where problems are highlighted for refinements, and that it should not block users from submitting their feedback"*.

The suggestion of having the ACE as a validation step was also suggested in both sessions 2 and 5, where participants found it as a helpful way for providing feedback, as it provides guidelines of how a textual description could be structured and/or written. But still they did not want it to be an obligatory step, *"feedback submission should not be bound to writing in a specific way"* as they explained.

On the other hand in sessions 3 and 4, the end-users in the sessions found it a necessary tool for providing feedback. Both participants during the session verbally discussed their problems (i.e. problem related to the fictional scenario provided) easily, however when it came to expressing it as a written description, it wasn't an easy task. They didn't know *"how to start"*, *"how to write things correctly"* to be *"clear, meaningful and easily understood"*. Thus, they found ACE as an *"effective tool for providing guides and instructions on how to structure text"*, especially that the toolbox provided examples for each sentence structure.

In conclusion, ACE utilization was positively received by all participants, but they had different perceptions and conditions on how it should be used as discussed in this section. This was included in an initial architectural design for structured feedback modelling (Sherief, Abdelmoez et al. 2015), which was validated in this study. The architecture was redesigned as shown in Figure 42 to switch the phases of **Feedback Structure Validation**, and **Sentence Structure Validation**, so that the former comes first in the design, to ensure the latter is an optional stage put for refinements, before the actual storage of the feedback in the knowledge base. However, given there is a tool (i.e. in future work) that utilizes such concept it would be left for users to write using ACE from the beginning or not, and in all cases it will be validated before actual storage. Furthermore, the architectural design was modified to show that the ontology knowledge base stores the feedback types' classification,

the rules that govern their usage and the final validated instances. Also, an input flow from the ontology to the reasoner was added to show that it uses the defined structures and rules to validate the end-user feedback. And finally, shows that the engineer role also uses the ontology knowledge base to read the structured feedback instances and use it in the evolution process tasks as explained in section 6.3.1.

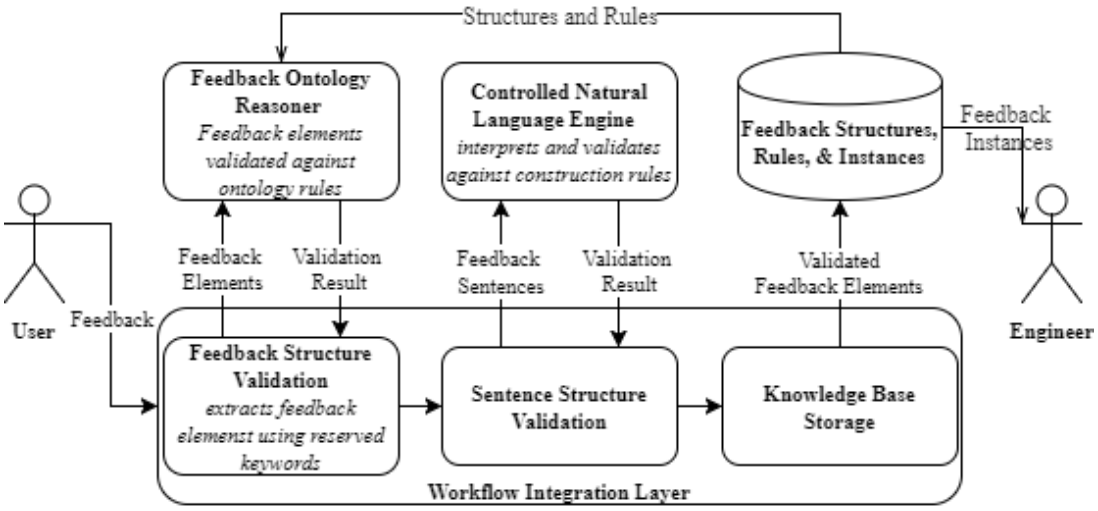

Figure 42. An Architectural Design for Structured Feedback Modelling

Concerning the Feedback Types and their components, during the design sessions whenever the participants needed to provide a feedback they mapped the input they wanted to give to the feedback definitions already existing in our classification. If it mapped to one of the feedback types, then they started analysing whether its components were suitable for providing useful descriptions or not. On the other hand, if it didn't map to a feedback type, then adding a new type was discussed and designed.

It is worthy to mention that before the PD study, feedback types were defined using a set of rules derived from the forums analysis study and formalized using ontological definitions. These definitions enabled the users to enter their feedbacks, which were mapped to the rules in order to identify the feedback type and therefore validate its structure. This initial way of working with feedback was based on usage of feedback in online forums where users may not tend to specify the types of feedback they are giving, especially that there is no process to ensure how users communicate with the support teams of large software suites developed for mass usage. Therefore, our idea was to provide simple structures that users could use to provide useful descriptions, and ensure minimum mandatory content.

However, this research has evolved after the conduction of the interviews study with software engineers towards applying the feedback structures in business context of software companies that develop customized software to their customers. These customers tend to be more motivated, and methodical practices could be followed. This resulted in a

change in the perspective in which feedback types are handled. Instead of using the ontology to detect the feedback type and validate it, it will be used as a flexible and extendible tool for feedback structuring, validation and storage (i.e. users understand the feedback type's meaning and usage and choose to provide a certain feedback according to the situation and need). Using ontologies also enables utilization of ACE plugins easily, and further querying of the stored feedback threads as discussed in the following section.

Consequently, in the PD study the templates were designed that conform to the rules that were previously designed, but a toolbox with the types of level of details was provided for flexible customization of each feedback type. It is the ontology's role to ensure that the minimum amount of information is met, and that the extended components (if any) are not disallowed for that feedback type, and store it in a structured interrelated manner for further querying.

Below is a detailed list of the comments and recommendations made for each feedback type to show how the feedback types' updates were derived from participants' quotes:

Topic Definition: this feedback type was detailed by **Explanation** and **Task Context**. In all design sessions both types of participants agreed that these details are enough as initial mandatory information. They did not want to add extra mandatory levels of detail in order not to hinder the end-users' experience. Also, extra components are dependent on the problem and therefore may vary from one case to another. Moreover, they can be systematically investigated by engineers when needed, and linked to the feedback thread.

Since Variations are possible, an example that occurred in one of the design sessions is that engineers recommended Social Context as an essential component in the topic definition *"especially in new features and enhancements, because it helps understand the customers' roles, restrictions, with whom and how they interact, which are essential information that directly affect the solution design and requirement fulfilment"*. So the end-user dragged the component from the toolbox and dropped it in the feedback template and described it. In this case the Topic Definition was detailed by Explanation, Task Context, and Social Context.

Therefore, the specified levels of detail are the mandatory ones, but any other level of detail can be associated for a richer feedback, unless explicitly specified that they could not be used.

Investigations: This feedback type allows engineers to write a textual question about the level of detail they want to inquire about. The template they were given allowed the engineers to ask using any level of detail except **Concise** and **Feature Definition** that were disabled in the toolbox as shown in Figure 43. However, during the PD design sessions,

engineers recommended that they should be enabled especially feature definitions when adding new features on enhancements.

For example, given a topic definition that the end-user explained, and a feedback elaboration that she provided afterwards, the engineer gave an explanation for her understanding of the new requirement and asked a **Concise Question** *"Can you please confirm that this is the requirement scope?"*

Another Example, the engineer needed to ask about a specific aspect of a new feature, so she sent an investigation template to request more information about that aspect in the **Feature's Definition** *"I'm concerned with the accessibility of this feature. Will the staff members be able to clone formats from only their own courses or everyone's?"*

Figure 43. A Level of Detail Toolbox showing disabled content

Therefore, the Investigation rule will be modified to include any possible inquiry on one or more type of **Levels of detail**. Also, it is worth to note that Investigations may include other level of details besides the investigated one. For example, an engineer may provide explanation of his understanding of the requirement and then ask question(s).

Addition: as explained in section 6.3.1.1.3 that this type emerged during the design sessions. By definition an Addition occurs when the engineer clarifies that the reported problem is a help request and that there exists a feature that resolves that issue (using the feedback type Problem Correction). However, the end-user decides to modify this feature definition's scope, for example, by enhancing its accessibility (which happened in design session 4).

Therefore, the Addition feedback type will be modified accordingly, as it always means an enhancement to an existing feature. So the minimum needed mandatory information is the new **Feature Definition**, and references a **Problem Correction** feedback type.

Mitigation: In our definition the mitigation feedback type means providing a solution to the topic definition on which the thread is held. In the design sessions this feedback type was logically divided into two different subtypes: **Solution** and **Proposal**.

All participants confirmed that the **Scenario** is a mandatory level of detail for both types (Solution and Proposal) as it will be used a baseline for the next situations that could occur. For example, a "*usage scenario in a Proposal can be used as it is in the Solution if the proposal was accepted"*. Moreover, **Exemplifications** were confirmed as a mandatory component in both Proposal and Solution Templates, and engineers confirmed that "*it is helpful especially if there are changes in the UI that requires screenshots"*. They also added that they would like to ensure that *"the scenario is given in steps in not as textual paragraphs"*. This would *"improve the readability of the solutions, and reduce the effort that end-users will spend in applying them"*. Also, to "*help link each step (if needed) to other feature(s) that it utilizes, which will help maintain the feature model"*, as mentioned in the feedback acquisition and communication process updates.

The difference between them is a logical difference that is clarified in the communication process between engineers and end-users explained in section 6.3.1.2, where in some straight forward cases of bug fixes there is no need for proposals (so it could be skipped). While, in other cases it could be used to make agreements with end-users on a planned solution, so that no effort is spent on an implementation that could be rejected.

Therefore, a new feedback type Proposal will be added that carries the same definition as the Solution. Both will be subtypes of the Mitigation feedback type as shown in Figure 44.

Problem Extension: is a feedback type used after trying a solution that partially resolved the issue. This type was originally detailed by **Explanation** and **Environmental Context** in the provided template. However, both engineer and end-user participants agreed that *"Explanation is best level of detail to be used to describe the new emerged problem, same as used with Topic Definitions"*. However, there is no need to mention the Task Context as in the Topic Definition, because this could be deduced from the feedback thread, but if needed it could be specified explicitly. Also, in both scenarios used in the PD design sessions there was no need for the Environmental Context information, and therefore it was recommended to be left optional for end-users to add according to the case.

Therefore, the final rule definition for Problem Extension is detailed by **Explanation**, and references Mitigation, or Mitigation Corrections. Also, the distributed template mandated this type to be accompanied by a **Confirmation** that references Mitigation and another **Confirmation** that references the Problem from which the extension evolved. This is to confirm that the provided solution partially resolved some of the issue, and to confirm that the problem still exists and did not close.

Mitigation Trial Failure: is a feedback type used after failing to try the provided Solution. This type was originally detailed **Trial** and **Environmental Context**. However, engineers pointed out that only the Trials that the customer made should be mandatory information to show the steps that the customer made to try the usage scenario provided in the Solution "*It helps the engineer to walkthrough the steps the customer made to further understand the problem"*. Moreover, in both scenarios used in the PD design sessions there was no need for the Environmental Context information, and therefore it was recommended to be left optional for end-users to add according to the case.

Therefore, the final rule definition for Mitigation Trial Failure is detailed by **Trial**, and references Mitigation, or Mitigation Corrections. Also, the distributed template mandated this type to be accompanied by a **Negation** that references Mitigation and another **Confirmation** that references the Problem from which the extension evolved. This is to negate that the provided solution resolved the issue, and to confirm that the problem still exists and did not close.

Negative Verification: this feedback type references Proposals feedback only to add **Negation** to them detailed by **Concise** or **Personal Context**, and provide justifications detailed by **Context** and **Explanations**.

As explained in section 6.3.1, end-users could reject a proposal and end the communication by providing Negative Verification. Or, they can provide a negative verification explaining

why they do not agree on the proposal; however they still need a solution. So, they will explain their modification in a feedback Elaboration to be considered as new requirement information that should be considered. This also, modifies the Feedback Elaboration rule to include the possibility to reference a Negative Verification.

Therefore, the rule definition for Negative Verification is to provide **Explanations** of why the proposal is not accepted, and using **Context** information to support that. Furthermore, a **Negation** is associated with that type to reference the Proposal.

Further enhancements to the levels of detail is that engineer participants suggested that each level of detail can be explained using multiple methods. For example, and Problem Correction feedback type is detailed by Feature Definition *"that could be explained both textually and by providing a link to the specification document that contains that definition"*. Also, Question is a specific method that is used only with Investigation feedback types. Investigations should contain at least one Question.

Figure 44 represents the final classification of feedback types after the PD study. Also, Table 18 summarizes the components updates and put the all the feedback types in their finalized form:

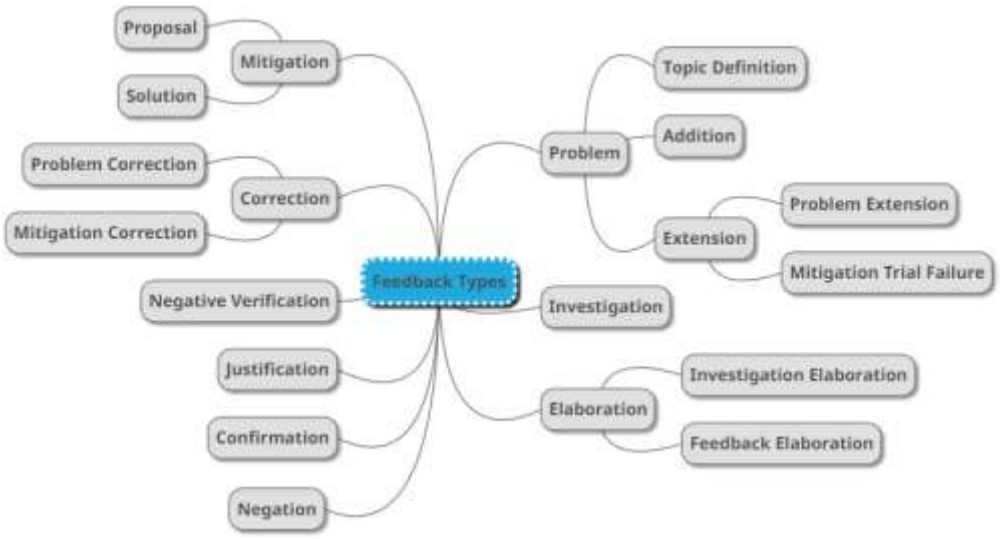

Figure 44. The Participatory Design's Study Final Classification of Feedback Types

Table 18. A Finalized List of Feedback Types and their Components

Feedback Type	Rule
Topic Definition	Detailed by Explanation
	Detailed by Task
Investigation	Detailed by Level of Detail that is Described by Question
	References Type(s) Problem , Elaboration, Negative Verification

Investigation Elaboration	Detailed by Level of Detail
	References Type Investigation
Feedback Elaboration	Detailed by Level of Detail Except Concise
	References Type(s) Problem, Negative Verification
Problem Correction	Detailed by Feature Definition
	Detailed by Explanation
	References Type(s) Topic Definition, Problem Extension, Addition
Addition	Detailed by Feature Definition
	References Type(s) Problem Correction
Mitigation (Proposal & Solution)	Detailed by Scenario that is illustrated by Scenario Step(s)
	Detailed by Exemplification
	References Type(s) Topic Definition, Addition, Problem Extension
Mitigation Correction	Detailed by Scenario that is illustrated by Scenario Step(s)
	Detailed by Explanation
	References Type(s) Solution AND Extension
Problem Extension	Detailed by Explanation
	Has Type Confirmation References Type Solution
	Has Type Confirmation References Type Problem
	References Type(s) Solution, Mitigation Correction
Mitigation Trial Failure	Detailed by Trial
	Has Type Negation References Type Solution
	Has Type Confirmation References Type Problem
	References Type(s) Solution, Mitigation Correction
Negative Verification	Detailed by Context
	Detailed by Explanation except Concise, Usage Scenario
	Has Type Negation References Type Proposal
Confirmation	Detailed by Concise or Personal Context ONLY that is Described by Text ONLY
	References Type(s) Mitigation, Correction
Negation	Detailed by Concise or Personal Context ONLY that is Described by Text ONLY
	References Type(s) Mitigation, Correction

Finally, all these feedback types' modifications will be improved in the ontology classification and rules definitions (explained in details in the next chapter). This is to provide formalism to our research. Formal methods helps us to avoid overlooking critical issues, provides a standard means to record various assumptions and decisions, and forms a basis for consistency among many related activities. Also, this will validate the architecture in Figure

42 that was proposed in (Sherief, Abdelmoez et al. 2015) and help demonstrate the integration of its components together, and the outcomes they produce, which are essential inputs to the feature specification extraction process explained in the next section.

6.3.3 Feature Model and Feature Specification Evolution process

This section explains the results regarding the issues related with features specification documentation. As discussed with the participants in the introductory sessions that there are two main issues that are focused on regarding the requirements documentation. The first problem is related to the description's content, while the other is related to the lack of systemized process for extracting information and keeping the documentation up-to-date.

The first problem is prompted, because requirements gathered during the maintenance phase are typically gathered through unstructured end-users' feedback and a series of unguided communication between both the end-users and the engineers. This causes information loss, because there are no guidelines for how this communication should happen and how information exchange during this communication should be documented. Therefore, the task of updating the documentation and putting important details into consideration is left untraced, as it is then very hard to remember all the detailed descriptions when documenting feature specification updates. So it would be left to the engineers' understanding and experience to write down the documentation, which may in turn contain a great deal subjectivity.

Furthermore, the second problem mainly occurs in the maintenance phase where there is an initial specification document that is not updated with each issue reported by the customer (de Souza, Anquetil et al. 2005, Kajko-Mattsson 2005, Leotta, Ricca et al. 2013). As gathered from the interviews study conducted earlier in this research, is that engineers are not keen on that task because of the effort and time it takes. Also, the lack of facilitating methods and guidelines for the documentation process is what makes it an ad-hoc task that is difficult to manage.

Thus it was our aim in this study is design with the engineers 1) a more detailed structure for the feature specification documentation; 2) how feedback structures could be used as a validated and formal source to provide input for the documentation; 3) what is the main steps that the engineers could follow to provide a systematic means for the information extraction and documentation process.

In the PD design sessions, engineers were given a draft for the feature specification template shown in Figure 15, along with an example of a feedback thread shown in Figure 39 to help them visualize how feedback could be related to form a thread of interrelated information. Also, the communication flow during the session with end-user using the design

feedback templates gave them an idea about the content available in each feedback in the thread.

Designing how the description could be extended with more details containing more sections for organization, each engineer provided insights regarding how to divide the description section. Each one provided advice according to his opinion, experience and preference. The common two main sections were the **Description** section and the **Scenario** Section. The content of the Scenario section can be extracted from the feedback thread as *"we have a usage scenario for each Solution and if this scenario was updated for example in a Mitigation Correction, then it could be updated accordingly"*.

On the other hand participants agreed to keep the Description section as it is, but add subsections with more precise levels of detail according to the case. For example, there are engineers who mentioned that context information is important, but *"we cannot be sure which type we may need, in enhancements and new feature, usually social context information is essential, while in bug fixes we might need task context or environmental context."* So it was concluded that it should be left to the engineers to extract the needed information from the feedback thread, but provide the suitable guidelines and tools to inform that task.

Thus, the need for designing such process was triggered. Having the stored feedback thread engineers suggested that they *"could retrieve it and view it"* in order to start *"marking the suitable Levels of Detail (LODs)"* that will be used to update the existing Feature Specification Document (FSD). Given that in this a definition of the classification of feedback types and their components in ontology was developed. Thus, the actual instances of feedback will be stored in the Ontology knowledge base and therefore can be queried using SPARQL querying language that could retrieve the needed information for such purpose.

In Figure 40 the last sub-process "Update the Feature Model and Feature Specification Document" is triggered after the customer provides a Confirmation either after trying the solution that successfully resolved his issue, or after receiving a Problem Correction for the help request or issue he has raised. When a confirmation is received after a successful solution trial the engineer should go through a process of documenting the modifications made to the related feature and update the feature model according to the scenario provided in the solution. Also, when a confirmation is received after a problem correction the engineer may need to update the feature description in the feature's documentation in order to cover the gaps in the description that led to the customer's misunderstanding. This could be directly extracted from the new feature definition with detailed description that was provided in the feedback description.

The sub process in Figure 40 is detailed in Figure 45 where all the steps needed to update the documentation and feature model are detailed. The engineer starts by **"Marking the**

Levels of detail (LODs) that he needs to update the documentation with". A discussion was held with the participants on the feedback types from which information could be extracted which is summarized in Table 19 below. The types they referred to as the allowed list of feedback types to extract from was: Topic Definitions, Addition, Elaborations, Mitigations, and Corrections. On one hand, they supported with argumentation why each type could be needed. *"Topic Definitions and Additions contain important explanations for features and task context information that can be used to provide descriptions of the feature usage in certain contexts"*. Moreover, *"Elaborations whether they were answers to questions or provided by end-user to complete important information can be used to complete the descriptions of features like: feature definitions, contextual information, exemplifications".* Additionally, *"Mitigations are the main source for describing the solution through usage scenarios, while Mitigation Corrections (if any) can update that description to ensure updated solution are stored".* Finally, *"Problem Corrections can be used to re-define or refine feature definitions, especially after being misunderstood by end-users".*

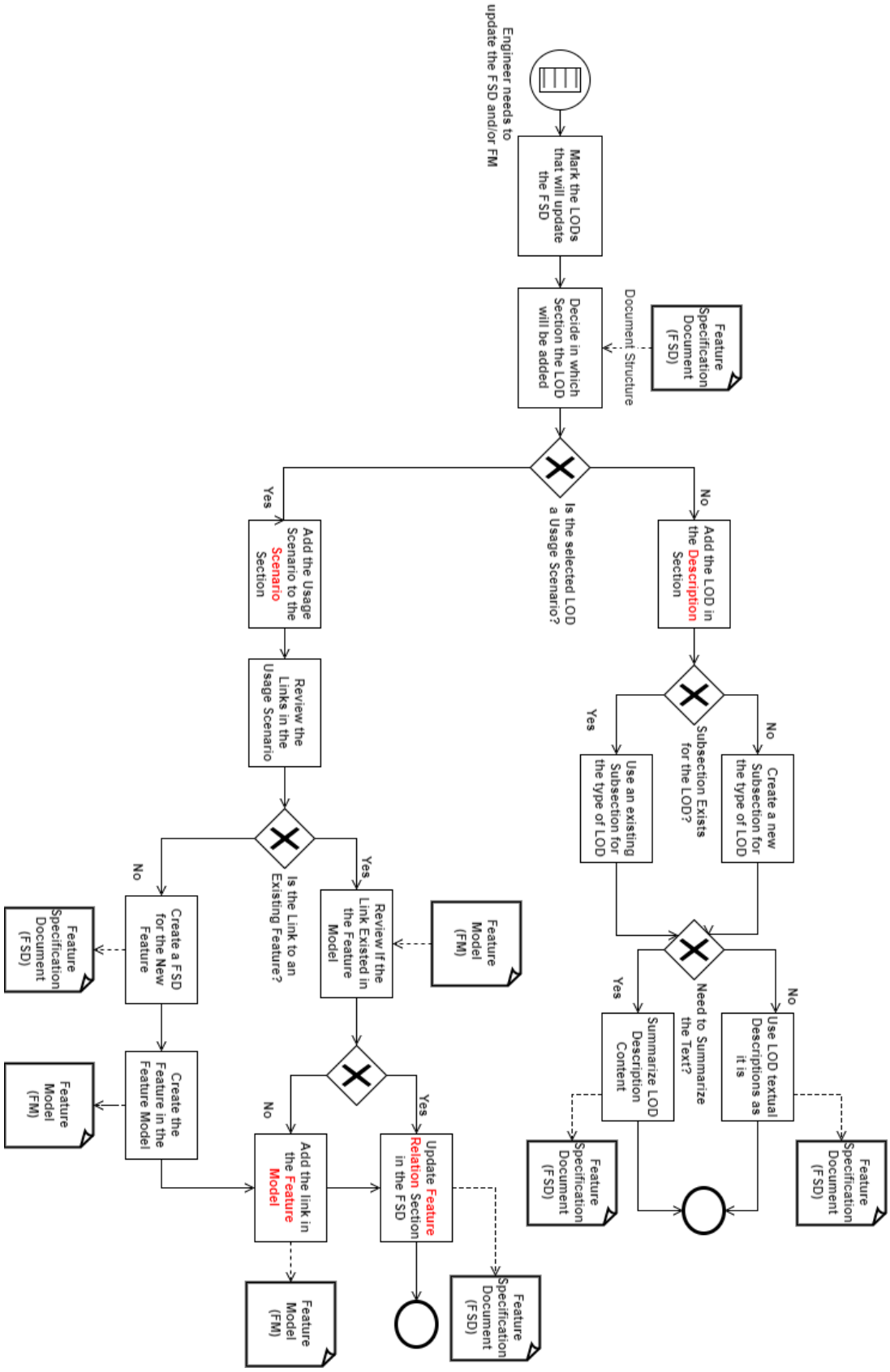

Figure 45. The Feature Specification and Feature Model Update Process

On the other hand, there were feedback types that were used during the session that were found unnecessary to use during documentation update such as Investigations, Mitigation Trial Failures, and Problem Extensions. They argued that *"there is no need to store the questions asked instead the answers provided. Storing questions in the document will make it look like a dialogue which is not needed in documentations' updates that will be used as a reference in future updates"*. Also, *"Mitigation Trial Failures and Problem Extensions report the problems in the solution, while only documentations of working solutions are needed in their final version"*, and after these two types normally engineers are requested to provide Mitigation Correction, which can be used to fulfil such purpose as mentioned above.

Table 19. The Allowed List of Feedback Types and Levels of Detail that can be used to fill the FSD Sections

FSD Section	Allowed Level of Detail	Allowed Feedback Types
Description	Context	Topic Definition, Addition, Elaboration (Investigation Elaboration, Feedback Elaboration), Problem Correction
	Any Depth except: Concise, Scenario	Topic Definition, Addition, Elaboration (Investigation Elaboration, Feedback Elaboration), Problem Correction
Scenario	Scenario	Mitigation (Proposal, Solution), Mitigation Correction
Feature Relation	Scenario Step containing Feature link	Mitigation (Proposal, Solution), Mitigation Correction

Afterwards, the engineer **"Decides in which section the selected LOD will be added"**. It is assumed that there exists an initial feature specification document (FSD) designed in the original project and that it contains the two main sections feature Description and Solution. Therefore, in the process shown in Figure 45, the engineer is asked in which section the level of detail he chose from the thread will be added. This step requires reading the document structure to identify the existing sections and subsections. If he marked that he would like to **"Add the LOD in the Description section"** then he would start deciding in which subsection inside the document the LOD will be added. After having the document structure read the next question is **"Whether a subsection inside the Description section exists or not"**. If yes, then the existing subsection will be used, but if no, then a subsection will be created. Next, the engineer will be asked whether he would like to insert the text from

the level of detail as it is or summarize and rephrase. Whatever the choice was the FSD will be updated and the process ends.

This process is iterative as indicated in the Figure 40; therefore, the same process in Figure 45 can be used again to update the different sections of the FSD, only it will follow a different path. Alternatively, if the selected LOD was a Usage scenario, then the only option is to **"Add the usage scenario in the Scenario section"** of the FSD. However, as mentioned in the sub process of documenting the proposal, solution, and/or mitigation correction usage scenario shown in Figure 41 that the scenario may contain links to features that are used to document how the feature model should be maintained. Therefore, in this process when the selected LOD is a usage scenario it should be **"Reviewed to identify if it contains feature links"** and use them in the actual documentation and updating the feature model.

When a link is found a scenario step then the next task is to **"Identify whether the link is to an existing feature"** or a new proposed feature. If the link is to an existing feature then the engineer will **"Review whether the link already existed in the feature model"** or not (i.e. between the feature in the scenario step and the feedback focus feature). If the link already existed, then **"the Feature Relation Section in the FSD will be updated"** with the feature name from the scenario step. If the link did not exist in the feature model then **"the feature model will be updated by adding the link"** specified in the scenario step between the two features, and then the affected features section will also be updated.

On the other hand, if the scenario step contained a link to a new feature that does not exist in the feature model. Then first **"a feature specification document for this new feature will be created"** and **"the feature will be created in the feature model"**. Then the feature model will be updated by linking this new feature to it using the link type specified in the scenario step. Finally, the affected features Section in the FSD will be updated with the new added feature. The final novel updated structure for the feature specification document is shown in Figure 46 below.

FEATURE NAME	
Priority:	Essential | Expected | Desired | Optional
Effort:	Months | Weeks | Days | Hours
Risk:	Dangerous | 3-Risks | 2-Risks | 1-Risk | Safe
Functional Area(s):	Word, Word, Word
Use Case(s):	UC-01
Description:	Subsections with Level of Detail Category Names; each subsection may contain a number of item(s) (i.e. paragraph(s)), For Example: Feature Definition: Exemplification: Environmental Context:
Scenario:	Feature Usage Scenario described in steps to organize information and browse each step individually.
Feature Relation:	Feature Name

Figure 46. A New Feature Specification Document Structure (amended from Robbins 2004)

6.4 Threats to Validity

There are three main threats to validity in this study:

1) In this study, the participants were given the classification of feedback structures, the deigned tool boxes that contained examples of the levels of detail usage in feedback, along with an initial draft to the feedback acquisition and communication method that they should follow in the maintenance phase. This could have influenced the quality of the participants' response especially in the area regarding the utilization of feedback structures and their components in the acquisition and communication process. For example, they could have follow or adopt to a large extent the ideas, information, and process paths represented by the examples and models. To minimize this effect, the study moderator constantly advised the participants to think out of box and generate their own ideas. As a proof of that is the new feedback types that evolved from the sessions, and the new paths that evolved the initial method draft to its final version.
2) The time limit given to the participants was tight and could affect the quality of their performance as raised by some of them. However, highly experienced engineer participants felt comfortable with the time limit and believed that it would not cause any harmful effect to the quality their outcome. However, in two sessions the end-user role participants took more than the specified time to the tasks (specifically the feedback acquisition tasks), but they and the engineer participants of these sessions were flexible with the duration and agreed to stay longer.
3) Some of the end-user roles participants involved in the study actually had minor experience as software engineers (fresh-graduates). This could have affected how they involved in the process. For example, they sometimes forgot that they are involved as end-users and started discussing the modelling parts (RE models utilization and feature specification structure, requirements extraction process). To minimize this effect, the study moderator kept monitoring the discussion between the clients and the engineers, and emphasizing that each one should concentrate on playing his role.

6.5 Summary

In this chapter the fourth and final study conducted in this research was explained. From this study the main contributions of this research evolved. Before this study there was the classification of feedback and its constituents that were further confirmed and evolved with engineers. The engineer study highlighted the need for a new method of feedback acquisition and communication that could be used during the maintenance phase to inform their tasks and decisions. Thus an initial design was made. In this chapter, all the results were employed in practice that resulted in: a new updated list of feedback type, a new

method for feedback acquisition and communication method that also caters for the extraction and updating of requirements information. Finally, a new structure for the feature specification documentation was constructed in this study. In the next chapter the models designs will be explained. The models will be implemented using ontology structure.

7. Formalization of Feedback Structures and its Utilization

In the previous chapters, two user studies were presented; the second user study was conducted by analysing forums that contained real users' feedbacks on selected software products. Concepts evolved from the forums analysis and were represented as a thematic map that explained in details the concepts of a feedback structure. This was further validated from engineer's perspective. Additionally, a PD study was conducted to design with both the end-user and engineer a feedback acquisition and communication method that utilizes the developed feedback structures. Finally, a new engineering process was developed for updating the new structure for feature specifications and software's feature models after changes that occur in the maintenance phase.

In this chapter, an ontology design that employs the reached results from the previous studies will be developed. The concepts will be arranged in a taxonomic (subclass–superclass) hierarchy, defining the object properties and the restrictions for each property, and providing formal definitions (i.e. rules) for each concept that governs its usage. This is to provide formalization for the results, to demonstrate how the feedback structures and threads resulting from the communication are stored as instances in the ontology's knowledge base. Also, to support the developed theoretical concepts with the possibility for automated reasoning through implementing a running demonstration that illustrates the utilization of feedback structures in the updating of feature models and the new structure of feature specification.

7.1 Introduction

Ontologies define a common vocabulary for researchers who need to share information in a domain (Noy and McGuinness 2001). It includes machine-interpretable definitions of basic concepts in the domain and relationships among them. Our objective is to build ontology of feedback concepts in order to reach a common definition of the structure of feedback and the rules and relationships that govern its use. Also, to define the structures of feature models (that already exist in the literature), and feature specifications that was renovated in this research to support the engineering process for feedback acquisition, communication, and keeping requirements information up-to-date during the maintenance phase.

The reasons why it is needed to develop the ontology are (Noy and McGuinness 2001):

- **To share common understanding of the concepts** that constitutes the structure of feedback. This will enable users to describe their feedback into discrete and well defined pieces that can be understood easily and thus makes the feedback more meaningful, useful, and manageable for further knowledge extraction. Moreover, a well-defined structure eliminates subjectivity in interpreting users' feedback, as one

analyst may interpret it in a way, while another one may expose a totally different interpretation.

- **To make domain assumptions explicit.** Specifications of domain knowledge are useful for new users who must learn what terms in the domain mean. Also, this way they are easier to validate and change. Also, they help to join information that normally resides isolated in several separate component descriptions, and it provides background knowledge that allows non-experts to query from their point of view.
- **Analysing domain knowledge is possible** once a declarative specification of the terms is available. Formal analysis of terms is extremely valuable when both attempting to reuse the ontologies and expanding them. Often the ontology of the domain is not an objective in itself. Developing ontology is like defining a set of data and their structure for other platforms to use. In our research the ontology will enable the formal definition, validation and storage of formal feedback instances that could be further queried to provide the needed inputs for the engineering process developed to aid engineers in keeping requirements information and models up-to-date.
- **To enable reuse of domain knowledge.** This is a very important advantage in building ontologies. The ontology when filled with user feedback instances will create a knowledge base that can be utilized for other tasks. For example, evaluation knowledge in the knowledge base can be reused to recommend to a user that they may try a suggestion/ solution of other users in a way similar to collaborative filtering.

In general there is no definitive correct way for building ontologies. The ontology is a model of reality of the world, and the concepts in the ontology must imitate this reality. After an initial version of the ontology is designed, it can be evaluate and debugged by using it in applications or problem-solving methods or by discussing it with experts in the field, or both. As a result, the initial ontology will almost certainly need to be revised. This process of iterative design will likely continue through the entire lifecycle of the ontology.

In this research, after the initial version of the ontology was developed (in the forums analysis study) the feedback structures rules were discussed with engineers in the interviews study. That was to validate it from their industrial perspective and their needs, and also to gain knowledge about how it could be utilized to help them during the tasks they perform in the maintenance phase.

This process of iterative design was continued through the entire lifecycle of the ontology development in this research. The final version of the ontology design was reached following the PD study where participants have validated the feedback structures and practically used them in designing the feedback acquisition, communication, and

requirements' updating. To further validate the ontology implementation, this application was tested using instances of feedback threads that took place in the PD design sessions.

7.2 The Ontology Development Process

In the ontology development the seven step process defined in (Noy and McGuinness 2001) will be followed. The process is summarized as follows:

1) Determine the domain and scope of the ontology
2) Consider reusing existing ontologies
3) Enumerate important terms in the ontology
4) Define the classes and the class hierarchy
5) Define the properties of classes—slots
6) Define the facets of the slots
7) Create instances

First step is to start defining the domain and scope of the ontology. To start, one way to define the scope of the ontology is by defining a set of competency questions. These are the questions that the knowledge base based on the ontology should be able to answer. These questions will be used to make a judgment about whether the ontology is acceptable: Does the ontology contain enough information to answer these types of questions? Do the answers require a particular level of detail or representation of a particular area?

In the domain of User-Driven Feedback Modelling for Supporting Software Evolution, the following are possible initial set of competency questions (Noy and McGuinness 2001):

1) What are the mandatory attributes of a certain feedback type?
2) What are the levels of details that can be used for each feedback type?
3) What are the methods that can be used for each level of detail when associated with feedback types?
4) Can a verification feedback type belong in the same feedback with a feedback type: Problem? (i.e. possible feedback types combinations in the same feedback)
5) Can a mitigation correction feedback type reference a feedback type: Problem? (i.e. possible relationships between feedback types belonging to different feedbacks)
6) Can a feedback that includes a mitigation feedback type reference another feedback that has type only Confirmation? (i.e. the existence of some types within feedbacks restricts their referencing to other feedbacks)
7) What ae the feedbacks related to the same feedback thread?
8) What are the referenced features in a problem?
9) What is the existing list of unresolved problems?
10) What is the existing list of mitigation trials on an unresolved problem?
11) What are newly introduced features to the feature model?

12) What are newly added relations between existing features in the feature model?
13) What are the levels of detail that can be used to update a feature specification?
14) What are the newly added sections/ items in a feature specification?

Being able to provide answers to these questions validates the ontology and formalizes the results from this research, which are: the definitions of feedback structures, the rules that govern their usage (the levels of details and methods that could be used), the possible referencing between feedback forming threads of communications, and finally extracting information from stored feedback types to update the feature models and feature specifications.

The second step is considering reusing present ontologies. It is always worth considering what other researchers and/or practitioners have done, and checking if it can be refined and extended for a particular domain and task. Reusing existing ontologies may be a requirement if the system needs to interact with other applications that have already committed to particular ontologies or controlled vocabularies. However, in this research there was no need for that because the foundation for this research is built on the fact that feedback are acquired and communicated in an ad-hoc manner, and therefore there does not exist any pre-defined structures or methods to define feedback components. Also, for the rest of the research, the outputs are based on the utilization of the newly developed feedback structures, and therefore there was no need for further utilization of any existing ontologies.

The ontology development process steps 3 to 6 (Noy and McGuinness 2001) are explained in detail in section 7.3, where: the important concepts were termed and explained, the classes and class hierarchies are defined, the object properties that link classes together are described, and finally the rules that governs each class usage and utilizes the object proprieties are explained.

The last step in the ontology development process which is creating the instances will be explained in detail in section 7.4, where feedback instances for one of the PD design sessions will be entered and validated through checking its conformance to the rules. Also an instance for the feature models for one of the scenarios used in the PD session will be entered in the ontology where the feature classes and object properties will be used to define the feature model hierarchy. The feedback thread and the feature model will be stored in the ontology knowledge base for further querying using SPARQL language to generate the necessary information needed to update the feature model and the feature specification.

7.3 The Ontology Design and Structure

In this section the design of the structure of the ontology will be explained. In the first subsection the 8 main classes of the ontology that were derived from the forums analysis and the participatory design studies that were conducted in this research will be described. Then, in the second subsection the object property hierarchy will be explained, which contains the different types of relationships between the classes in the ontology. Finally, in the third subsection the use of each class will be explained by defining the rules that classifies the class.

7.3.1 Class Hierarchy

Figure 47 shows the eight main classes of the ontology (Horridge, Knublauch et al. 2004). These classes are mainly derived from the results reached from the forums' analysis and the participatory design studies that were conducted and explained its results in the previous chapters (chapter 4 and 6). In this section, provides design descriptions for each class.

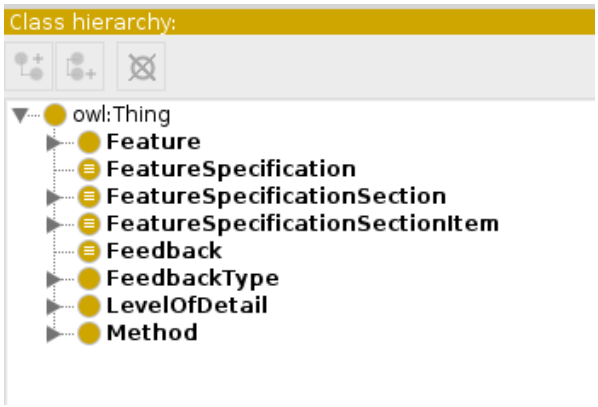

Figure 47. A Collapsed View of the Ontology Class Hierarchy.

7.3.1.1 Feedback Class

The **Feedback** class represents the single feedback that the users provide, which can be a start of a feedback thread representing the first problem in the thread or a response on others' feedbacks.

7.3.1.2 Feedback Type Class

A **Feedback** may contain one or more **Type** from the list shown in Figure 48. For example, a user can provide a feedback that contains only a single feedback type such as a problem that he wants to explain, or he can provide a feedback such as Mitigation Trial Failure that consist of multiple types such as Confirmation referencing Problems and Negations

referencing Mitigations. The hierarchy of this class and subclasses is derived from the novel classification of feedback types reached from the forums' analysis explained in section 4.4.2, then validated and confirmed in the interviews study and the PD study. The final list of feedback type's definitions is explained in detail in section 6.3.2 and summarized in table 18. And thus their definitions will not be repeated again.

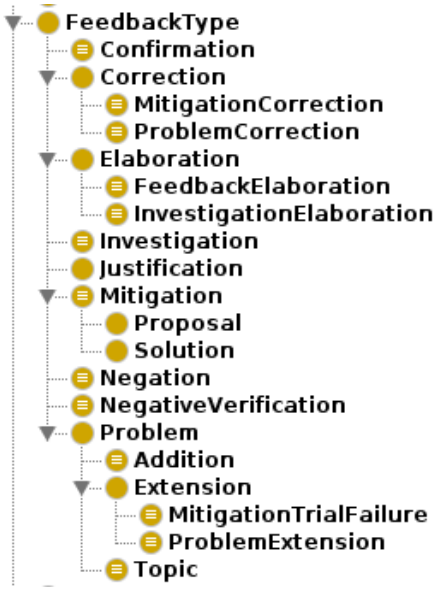

Figure 48. A Detailed View for the Feedback Types Class.

7.3.1.3 Level of Detail Class

The **Level of Detail** class represents the different kinds of details (i.e. feedback components) that the user can provide about the feedback type as shown in Figure 49. This class is divided into two subclasses of information that users provide: **Context** and **Depth**. The user can add one or more level of detail in his feedback type in order to better provide meaningful feedbacks. The subclasses of the context and Depth classes completely adhere to the classification concluded from the forums analysis study explained in section 4.4.3, and thus their definitions will not be repeated again.

However, in the PD study a new Depth type was designed with the participants named **Scenario Step**. This level of detail is a smaller grain than the Scenario, which is designed to hold each step of the scenario that the engineer provides in his Proposal, Solution, and Mitigation Correction Feedback Types. This will support the linking of each scenario step to the feature it utilizes whether this is an existing feature in the current specification, or a newly added feature suggested as part of the solution. Also, this will further enable the easy querying of these steps to get the information needed to update the feature model.

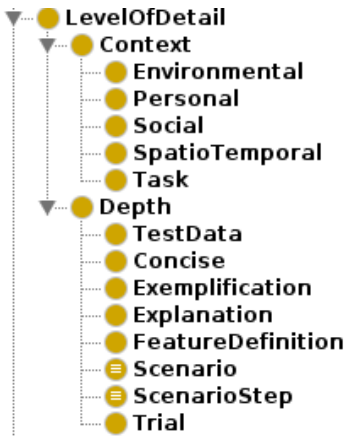

Figure 49. A Detailed View for the Level of Detail Subclasses

7.3.1.4 Method Class

The **Method** class provides the different ways of description that users can make use of while expressing their feedbacks, which is shown in Figure 50. These methods were initially defined as: text, snapshots, code snippets, or links. Details and explanation of the meaning of each method and its association with feedback types could be found in section 4.4.4. These methods were extended in the interviews study with the engineers, and the hierarchy was extended to include the File and Questions methods as explained in sections 5.7.1.1 and 5.7.1.3. Thus, their definitions will not be repeated again.

Still, the new subclass "**Feature Link**" that was devised during the participatory design study will be explained. As mentioned in the above section, a Feature link is a method used to describe if a scenario step needs to utilize other features than the subject feature of the feedback. Whether the link is to an existing feature in the feature model or a new proposed feature suggested as a part of the solution. In both cases a relation type between the two features will be specified (i.e. the subject feature of the feedback and the feature mentioned in the step).

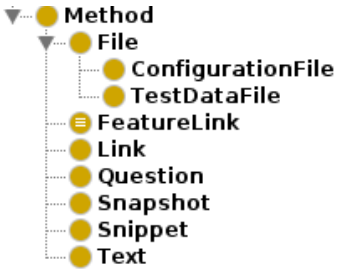

Figure 50. A Detailed View for the Methods Class.

The relationships between the four classes explained above can be described as: each feedback may contain one or more feedback types, where each type can be detailed by one or more context and/or depth categories, where each detail can be described by a method.

7.3.1.5 Feature Class

The **Feature** class, shown in Figure 51, serves two purposes. First, it is used to construct the feature model. In order to be able to represent feature models on the ontology, instances of the feature class must be defined, while the dependencies and cross-tree constraints will be defined as object properties connecting the features' instances together. Representing the feature model in the ontology will enable the engineers to utilize the features in their scenario descriptions, and update the feature model with newly added features and/or relationships. Afterwards, querying these scenarios will help keep the feature model updated.

Second, the feature class is also used to represent a feedback's subject feature. A subject feature is the problematic feature that the end-user reported an issue about. It is important that the feedback is related to a specific subject feature in order to help engineers narrow down the modification scope and accurately determine the impact of a change. Also, engineers will be able to utilize other features in their scenario steps using feature links.

There are two subclasses in the feature class, an Implemented Feature, and a Proposed Feature. The Implemented Feature represents features that exist in the feature model when the problem was reported. Therefore, when linking to an Implemented feature in the scenario step that is to emphasize or create the relationship between it and the subject feature.

In a scenario step engineers may link to a new proposed feature that does not exist in the feature model, because it was created as a part of the solution. This updates the feature model by adding a new feature and a new relationship with the subject feature. This distinction between the feature types is needed to identify how the feature model will be updated.

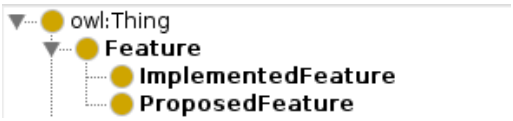

Figure 51. A Detailed View for the Feature Class

7.3.1.6 Feature Specification Class

The **Feature Specification** Class is used to define a new structure for documenting feature specifications that was devised from the PD Study and is shown in Figure 46. This structure combines several new sections that the engineers will use to update the document with the necessary feature information after each change. There are three types of sections, which are: 1) Description, 2) Scenario, and 3) Related Features, and they will be explained in the following section below.

7.3.1.7 Feature Specification Section Class

The **Feature Specification Section** class, shown in Figure 52, is the core component of the Feature Specification. The section types are: 1) the description section that is used to group levels of details that are used to describe the feature; 2) the Scenario section that is used to group the scenario levels of details, which is used to describe the usage of the feature in possible scenarios or tasks; 3) the Related Features section that is used to group the sub-features that are utilized by the scenario steps through feature links. Each feature link will be evaluated to see if the link existed or not and whether the link is to a new proposed feature or an existing implemented feature. Accordingly, the list of related feature to the main subject feature will be updated.

Figure 52. A Detailed View for the Feature Specification Section, and the Feature Specification Section Item Classes.

7.3.1.8 Feature Specification Section Item Class

This class represents the component that constitutes each section in the feature specification. Each item could include one or more levels of detail. It is the engineer's decision to choose the strategy he wishes to document the feature specification section with. One way, is to create an item for each level of detail in the description section, for example, creating an item that holds the feature's definition, an item for illustrating the UI exemplifications, or an item for contextual information that govern the feature's usage.

If the feature can have different usages when used in different tasks, therefore, the engineer will need to select the needed levels of details, for instance the feature definition, with its associated context information to be the item.

7.3.2 Object Properties

In this subsection the design of the object property hierarchy (Horridge, Knublauch et al. 2004) of the feedback structure ontology shown in Figure 53 will be explained. The classes alone will not provide enough information to answer the competency questions defined in section 7.2. Besides the taxonomy, a set of object properties where designed to describe the relation between the classes which are used to create the rules that governs the constitution of each class. Also, they are useful joins in query processing.

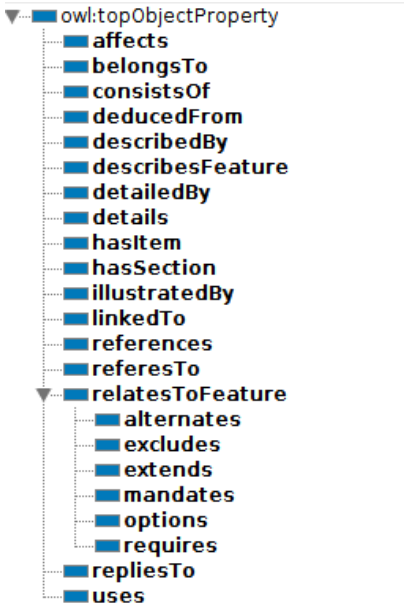

Figure 53. A Detailed View for the Object Properties Hierarchy

Web Ontology Language (OWL) Properties represent relationships between two classes. Properties may have a domain and a range specified. Properties link classes from the domain to classes from the range. It is important to realize that in OWL domains and ranges should not be viewed as constraints to be checked. They are used as `axioms' in reasoning. In table 20 below is a summary for the domain(s) and range for each object property defined in the ontology.

Table 20. A Summary of the Object Properties' Domain and Range

Object Property	Domain	Range
affects	Feedback	Feature
belongsTo	Feedback Type	Feedback
consistsOf	Feedback	FeedbackType
deducedFrom	FeatureRelationItem	ScenarioItem
describedBy	LevelOfDetail	Method
describesFeature	FeatureSpecification	Feature
detailedBy	FeedbackType	LevelOfDetail
details	LevelOfDetail	FeedbackType
hasItem	FeatureSpecificationSection	FeatureSpecificationSectionItem
hasSection	FeatureSpecification	FeaturespecificatioSection
illustratedBy	Scenario	ScenarioStep
linkedTo	LevelOfDetail	LevelOfDetail
references	FeedbackType	FeedbackType
relatesToFeature	Feature, FeatureLink	Feature

refersTo	FeatureSpecificationSectionItem	Level of Detail
repliesTo	Feedback	Feedback
uses	ScenarioStep	FeatureLink

Below is a list of examples representing the utilization of object properties as relationships between the domain and range classes. The examples are in the same order of the property list in table 20.

- A Feedback *affects* a Feature.
- A FeedbackType *belongsTo* a Feedback.
- A Feedback *consistsOf* FeedbackType(s). [consistsOf is *InverseOf* belongsTo]
- A FeatureRelationItem is *deducedfrom* a ScenarioItem.
- A LevelOfDetail is *describedBy* a Method.
- A FeatureSpecification *describes* a Feature.
- A FeedbcakType is *detailedBy* a LevelOfDetail.
- A LevelOfDetail *details* a FeedbackType. [details is *InverseOf* detailedBy]
- A FeatureSpecificationSection *hasItem* a FeatureSpecificationSectionItem.
- A FearureSpecification *hasSection* a FeatureSpecificationSection.
- A Scenario is *illustratedBy* ScenarioStep(s).
- A LevelOfDetail is *linkedTo* LevelOfDetail(s).
- A Feature or FeatureLink *relatesTo* a Feature.
- A FeedbackType references A FeedbackType
- A FeatureSpecificationSectionItem *refersTo* a LevelofDetail
- A Feedback *repliesTo* a Feedback.
- A ScenarioStep *uses* a FeatureLink.

The *relatesToFeature*, shown in Figure 49, is the object property that is used to: 1) relate features together to construct the feature model, or 2) relate a feature link to a feature in a scenario step to update the feature model by adding links and/or features.

This object property has 6 sub-properties that represent the relationships and cross-tree constraints in the feature model notation. The sub-properties have the same domain and range as the *relatesToFeature*. They are defined in table 21 as follows:

Table 21. The "relatesToFeature" Sub-Properties List and its Correspondence to the Feature Model Notation.

Sub-Property	Example	Correspondence to feature Model Notation
alternates	A Feature *alternates* Feature(s)	The **"Alternative (XOR)"** relationship between a parent feature and its child features (or sub features), where one of the sub-features must be selected.
excludes	A Feature *excludes* Feature(s)	A cross-tree constraint, where A and B cannot be part of the same product.
extends	A Feature *extends* Feature(s)	The **"Or"** relationship between a parent feature and its child features (or sub features), where at least one of the sub-features must be selected.
mandates	A Feature *mandates* Feature(s)	The **"Mandatory"** relationship between a parent feature and its child features (or sub features), where the child feature is required.
options	A Feature *options* Feature(s)	The **"Optional"** relationship between a parent feature and its child features (or sub features), where the child feature is optional.
requires	A Feature *requires* Feature(s)	A cross-tree constraint, where The selection of A in a product implies the selection of B.

7.3.3 Class Rules

After explaining the class and the object properties hierarchies, in this section their utilization in defining the rules (Horridge, Knublauch et al. 2004) that govern the class usage will be explained.

7.3.3.1 Feedback and Feedback Types' Rules

In this subsection a demonstration for the complete list of rules for the Feedback Class and all the Feedback Types sub-classes will be presented. Previously, in section 6.3.2 (in the participatory design study chapter) the final rules were explained and how the updates were derived from the participants' usage. The final list was summarized in table 18. Therefore, they will not be explained again in this section, only the screenshots showing the object properties and classes utilization will be presented in Figures 54-67. The implementation of

these rules in the ontology design will enable the actual realization of the feedback acquisition and communication engineering method shown in Figures 40.

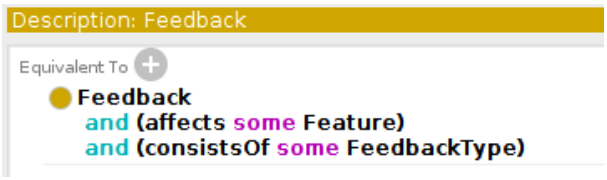

Figure 54. The Rule Description for the Feedback Class

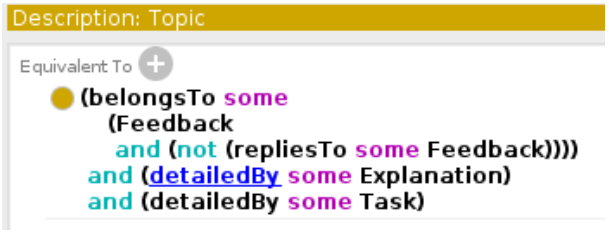

Figure 55. The Rule Description for the Topic Feedback Type

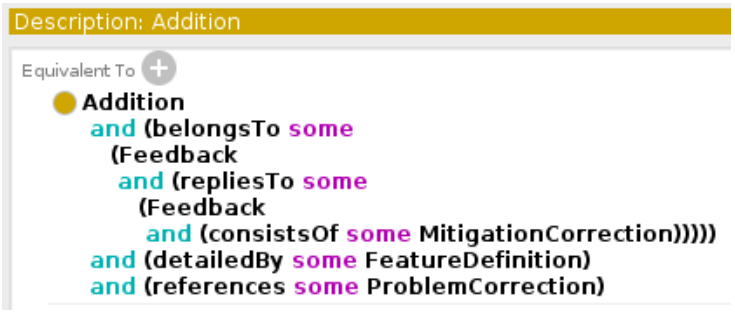

Figure 56. The Rule Description for the Addition Feedback Type

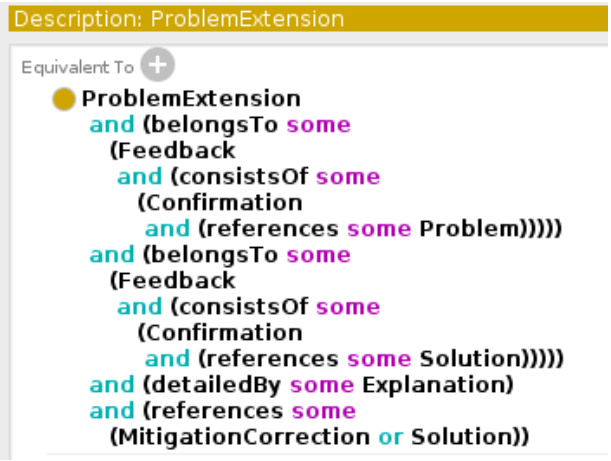

Figure 57. The Rule Description for the Problem Extension Feedback Type

Figure 58. The Rule Description for the Mitigation Trial Failure Feedback Type

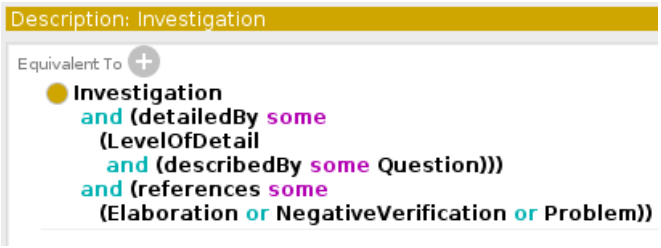

Figure 59. The Rule Description for the Investigation Feedback Type

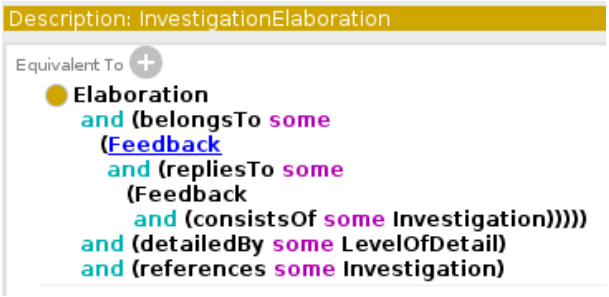

Figure 60. The Rule Description for the Investigation Elaboration Feedback Type

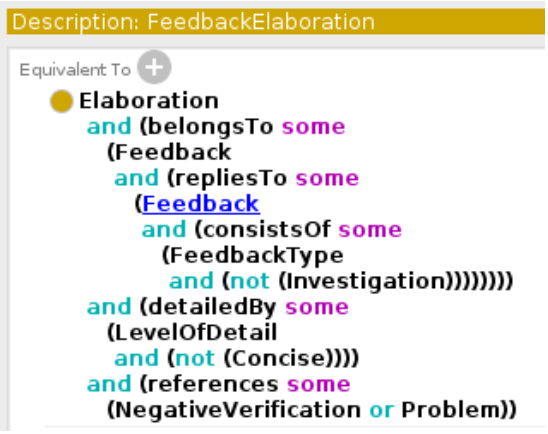

Figure 61. The Rule Description for the Feedback Elaboration Feedback Type

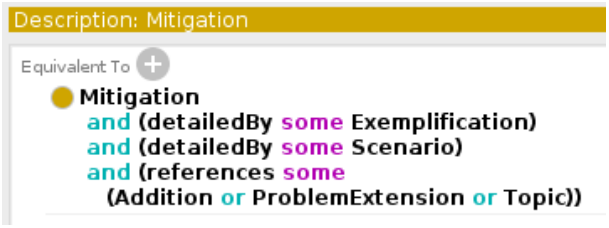

Figure 62. The Rule Description for the Mitigation (Proposal, Solution) Feedback Type

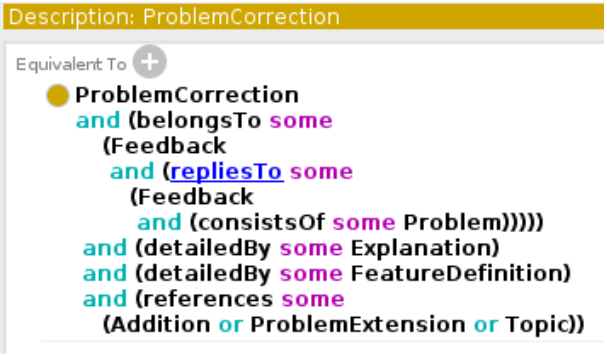

Figure 63. The Rule Description for the Problem Correction Feedback Type

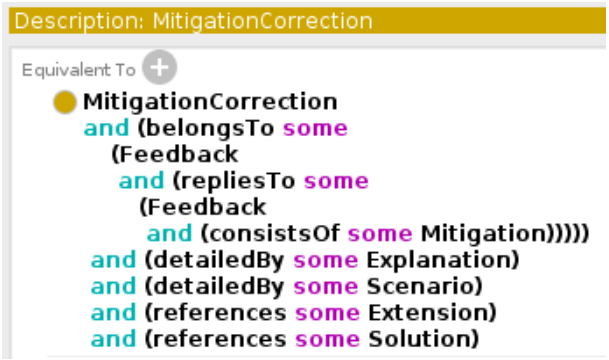

Figure 64. The Rule Description for the Mitigation Correction Feedback Type

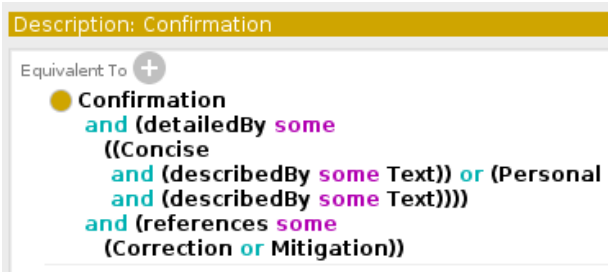

Figure 65. The Rule Description for the Confirmation Feedback Type

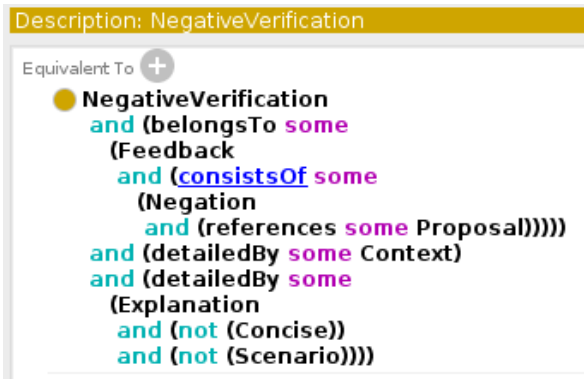

Figure 66. The Rule Description for the Negative Verification Feedback Type

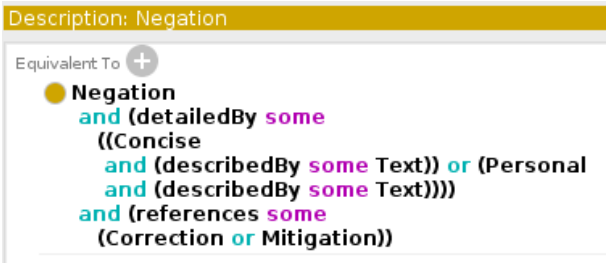

Figure 67. The Rule Description for the Negation Feedback Type

7.3.3.2 Methods and Level of Details Rules

In this subsection the explanation and rules for selected subclasses of Level of Detail and Method will be presented. A detailed process, shown in Figure 41, was designed in the participatory design study in order to enable the engineers to provide Scenarios in their Proposals, Solutions, or Mitigation Corrections in a systematic manner that enables them to further maintain the feature model and feature specification updated.

The Scenario, Scenario Step, and Feature Link sub-classes were selected as they directly affect the actual realization of the internal tasks detailed of the above mentioned process shown in Figure 41, and explained in section 6.3.1.2. The scope in this chapter is show the design that helped ensure the formalization of the designs reached in the previous studies. First, the Scenario Level of Detail rule in Figure 68 ensures that the scenario is illustrated by scenario steps. Second, each Scenario Step may or may not use some Feature Link Method in its description as shown in Figure 69. Finally, this Feature link (if used) must relate to some feature instance defined in the ontology knowledge base as shown in Figure 70.

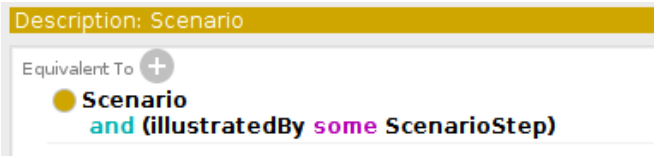

Figure 68. The Rule Description for the Scenario Level of Detail

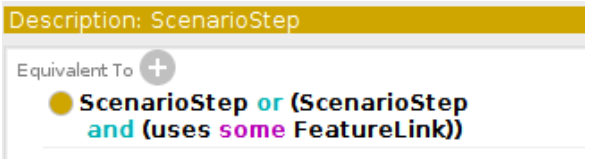

Figure 69. The Rule Description for the Scenario Step Level of Detail

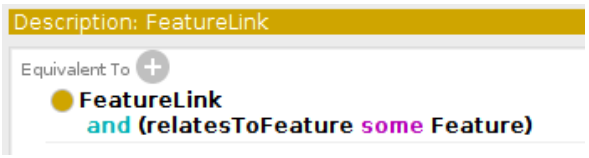

Figure 70. The Rule Description for the Feature Link Method

7.3.3.3 Feature Specification Rules

In this section the rules for all the classes and sub-classes hierarchy of the feature specification structure shown in Figure 46 is explained. This novel structure was developed with the engineers in the PD study. The following rules formalize the mentioned structure by defining each component of the feature specification, linking them together, and specifying how they are going to be filled.

In Figure 71, the basic rule for the Feature Specification is defined that ensure that it is linked to a specific feature (i.e. the subject feature of the feedback thread), and that it is composed of one or more Section as needed. In Figure 72, the feature specification section was defined as a class holding one or more item(s). The section can be a Description Section containing one or more items as specified in Figure 73, a Scenario Section holding one or more scenario items as specified in Figure 74, or a Feature Relation Section including one or more Feature Relation Items as specified in Figure 75.

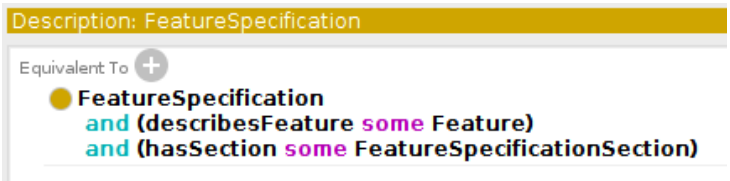

Figure 71. The Rule Description for the Feature Specification Class

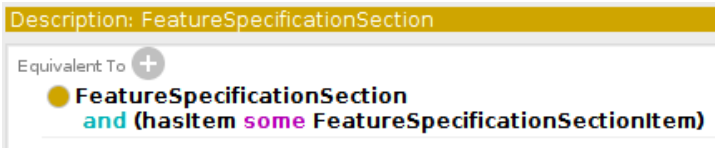

Figure 72. The Rule Description for the Feature Specification Section Class

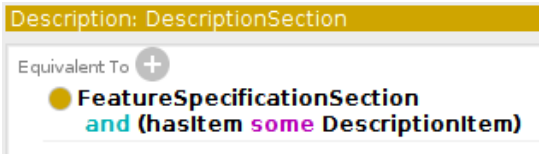

Figure 73. The Rule Description for the Feature Specification Description Section Sub-Class

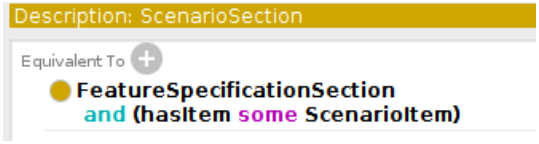

Figure 74. The Rule Description for the Feature Specification Scenario Section Sub-Class

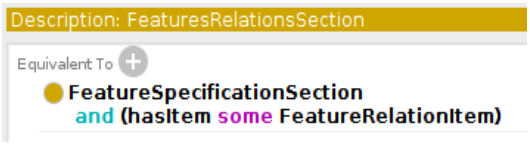

Figure 75. The Rule Description for the Feature Specification Feature Relation Section Sub-Class

Each type of Item is connected to one or more Level of Detail that the engineers can use to fill the feature specification sections as illustrated in Figure 72. Also, some of the feature section items can be deduced from another item as shown and explained in Figure 76. The set of allowed level of details was also designed with the engineers in the PD study and summarized in table 19 and explained in detail in section 6.3.3.

Figure 77 illustrates that engineers can compose the description section from items that refer to any type of level of detail belonging to a list of allowed feedback types as specified in the rule. Figure 78 demonstrates that engineers can only fill the Scenario Section with items connected to a Scenario Level of detail. Figure 79, demonstrates that the Related Features Section will be filled with items deduced from the Scenario Items that are connected to a Scenario Level of Detail that contains Scenario Steps Level of Detail, which uses the Feature Link Method. This also ensures that the Related Features Section is updated only if the Scenario Section was updated, as it links to the Scenario Item in the Scenario Section and not directly to the Scenario Level of detail.

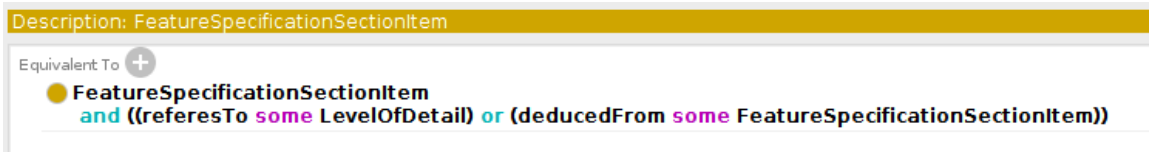

Figure 76. The Rule Description for the Feature Specification Section Item Class

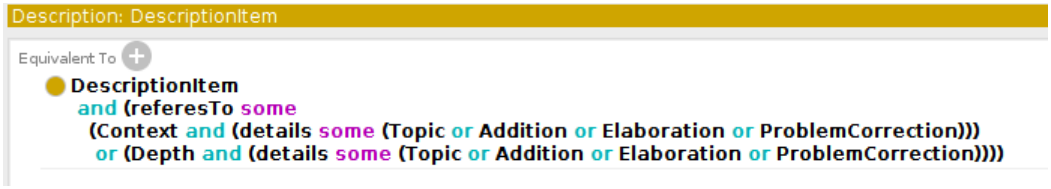

Figure 77. The Rule Description for the Feature Specification Section Description Item Sub-Class

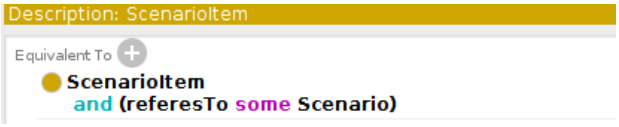

Figure 78. The Rule Description for the Feature Specification Section Scenario Item Sub-Class

Figure 79. The Rule Description for the Feature Specification Section Feature Relation Item Sub-Class

7.4 Explaining the Feature Specification Implementation

In this section a demonstration for the implementation of the feature specification will be explained through a running case. In order to realize this implementation there are important inputs that should be prepared first.

The first input is the feedback thread representing the communication between the end-user and the engineer. This thread will consist of several feedback instances. Each feedback consists of one or more feedback types that conform to the rues defined in the ontology.

The second input is the feature model's features and their relations. These features are also entered as instances and connected together using the object properties defined in the ontology. The initial feature model that is entered represents the current implementation of

the software system in use that will be utilized in the Scenarios entered by the engineers in the Mitigation or Mitigation Correction feedback types. Also, the feature model will be updated along the maintenance phase where new feature and relations will be added.

A group of queries were designed and implemented using SPARQL Protocol and RDF Query Language that are used to extract information from the feedback instances to update the feature model and the feature specification structure.

7.4.1 Sample Feedback Instances

In this subsection, the feedback thread and the feedback instances that constitute the thread will be illustrated showing their feedback types, levels of detail and methods used.

Figure 80 shows a feedback thread on the ontology, which is one the feedback threads that resulted from the participatory design sessions. The detailed session sequence showing the feedback types used is explained in section 6.3.1.1.1. The PD design sessions' threads were used as instances for the ontology design and testing, as they already conform to the rules of feedback types, which was one of the objectives of the study. Only, distinct samples from this thread will be shown in this section.

For simplification the sequence of the thread will be listed below:

- Feedback_1 is a Topic Definition feedback type provided by the end-user.
- Response_1_ Feedback1 is a Feedback Elaboration feedback type provided by the end-user.
- Response_2_ Feedback1 is an Investigation feedback type provided by the engineer to clarify some issues.
- Response_3_ Feedback1 is an Investigation Elaboration feedback type provided by the end-user that responds to the questions in the previous investigation.
- Response_4_ Feedback1 is a Feedback Elaboration feedback type provided by the end-user to add some requirements details to the enhancement.
- Response_5_ Feedback1 is an Investigation feedback type provided by the engineer to clarify some issues about the added details.
- Response_6_ Feedback1 is an Investigation Elaboration feedback type provided by the end-user that responds to the questions in the previous investigation.
- Response_7_ Feedback1 is a Proposal feedback type provided by the engineer.
- Response_8_ Feedback1 is a Confirmation feedback type provided by the end-user to accept the proposal.

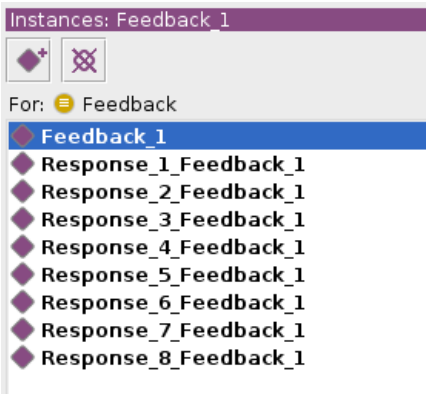

Figure 80. An illustration of a feedback thread on the ontology.

The end-user started the feedback thread by providing a Topic feedback. As shown in Figure 81, the Topic was described using the explanation and task context levels of detail, which are the mandatory components for providing this feedback type. Thus, this is a valid feedback, as it conforms to the rule defined by the ontology. Also, the end-user added an extra level of detail, which is personal context.

The feedback content descriptions are stored in each of these levels of detail so by clicking them, the engineer can view back the feedback content. Also, this will be explained in section 7.4.3 where the query results will be presented.

Figure 81. The Topic Feedback Type Used In Feedback_1 Showing the Levels of Detail Used In Its Definition.

Figure 82 shows Response_5 to Feedback_1 which was an Investigation feedback type reported by the engineer to clarify some issues in the elaborations provided by the end-user. The engineer asked two questions about the feature definition level of detail.

Figure 82. The Investigation Feedback Type Used in Response_5 to Feedback_1 Showing the Levels of Detail Used in Its Definition.

It is shown in Figure 83 that the end-user responded in the investigation elaboration with two feature definition levels of detail. This is because the rule for investigation elaboration restricts the end-users to provide answers that have the same type of level of detail

specified by the engineer in his investigation (i.e. question). This ensures that the end-user provides more specific and relevant answers.

Note: There is a naming convention for the instances for tracing purposes.

Figure 83. The Investigation Elaboration Feedback Type Used In Response_6 to Feedback_1 Showing the Levels Of Detail Used In Its Definition.

To further demonstrate how each answer is linked to one of the questions in the investigation. Figure 84 shows that the end-user provided an answer Feature_Defintition_4_Investigation_Elaboration_2, which the first response in the investigation elaboration in Figure 83, and linked it to Feature_Definition_2_Investigation_2, which is one of the investigation questions in Figure 82, and described it using textual method.

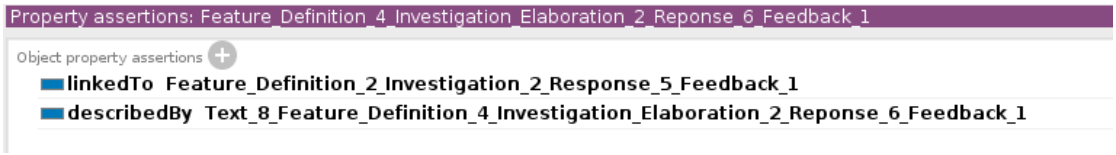

Figure 84. Investigation Elaborations Answer Showing its Link to a Specific Question in the Investigation.

Figure 85, illustrates the proposal that engineer provided after gathering all the necessary information needed to reach an initial plan and a solution design. The proposal was detailed using: 1) an exemplification that is described using a snapshot containing the UI updates; and 2) the engineer provided a scenario with the steps that the user should follow in order to use the new feature scenario. Figure 86 shows the scenario steps that illustrate the scenario provided in the proposal.

Figure 85. A Proposal Feedback Type and the Levels Of Detail Used to Describe it.

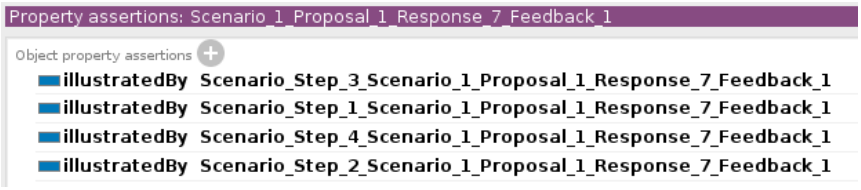

Figure 86. The Scenario Steps Level of Detail Used to Illustrate the Scenario Used in the Proposal Feedback Type.

Figure 87 show that scenario step 1 is described by two methods: textual description, and uses a feature in the description. Clicking on the utilized feature in the scenario step demonstrates how the feature link method is used to link the subject feature with a new proposed feature, and specifies the parental relationship between them as Optional as shown in Figure 88.

Figure 87. The Methods Used To Describe The Scenario Step and its Link to Another Feature.

Figure 88. The Relation Used In the Feature Link Method to Link to an Existing Feature in The Feature Model.

7.4.2 A Feature Model Instance

In this subsection the construction of the feature model will be demonstrated. Feature models are constructed using the feature class and the object property *relateToFeature* along with its sub-property list. First time the feature model is created all the features used will be instances of the sub-class Implemented Feature. During the maintenance phase any new feature added will be an instance of the sub-class Proposed Feature.

Figure 31 shown in section 6.2.2.2 demonstrates the feature model for the courses class used in the PD design session immersion scenarios. The feature model was designed in order to illustrate the implemented features hierarchy to the engineers to help them in the impact analysis task, and to design with them how the feature model could be updated after the requested enhancement.

Figure 89, illustrates the instance for the feature model in Figure 31. It shows that the Courses module has several sub-modules, where the relationship between the Courses and its sub-modules is a mandatory relationship. Therefore, the mandates object property was

used. Moreover, Figure 90 shows the “Adding new Course” sub-module and its related features, where “Bulk Course Creation” and “Course Template” are optional features, and “Define New Course” is a mandatory feature. Finally, Figure 91 illustrates the graph representation of the features’ instances, showing distinct colouring for the different object properties used (brown links for the mandates relation, and yellow links for the optional relation).

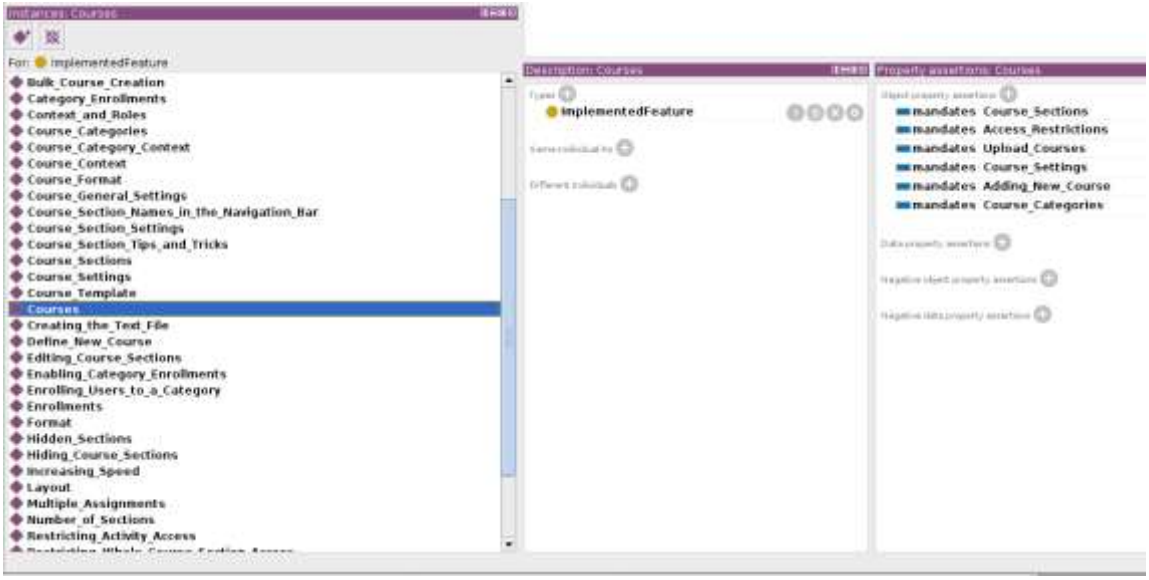

Figure 89. The Course’s Module Feature Model Instance

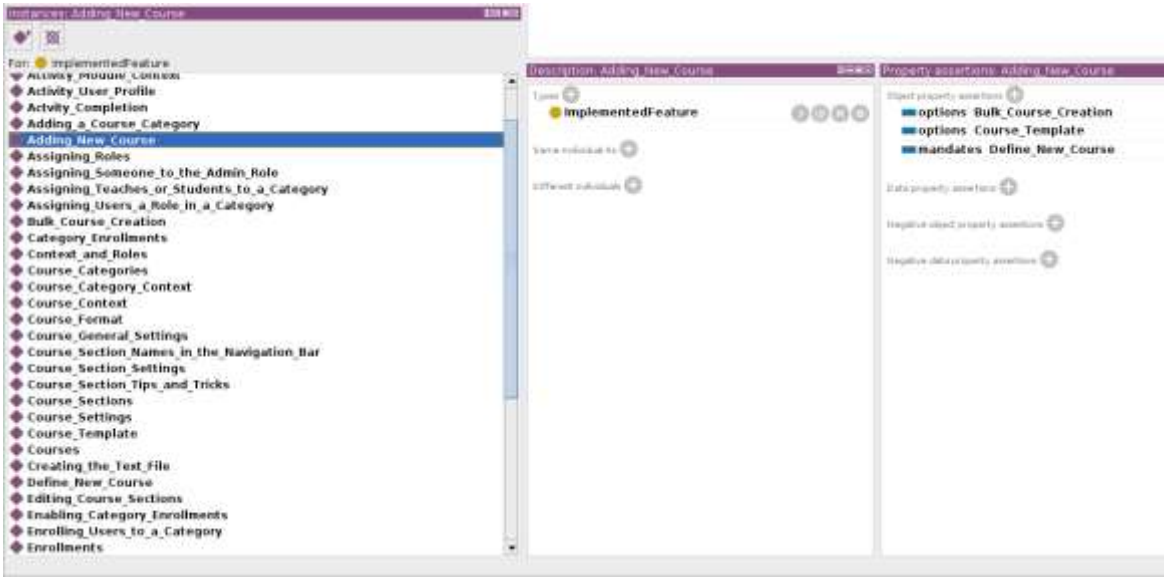

Figure 90. The Feature Model Instance for "Adding New Course" feature and its Sub-features.

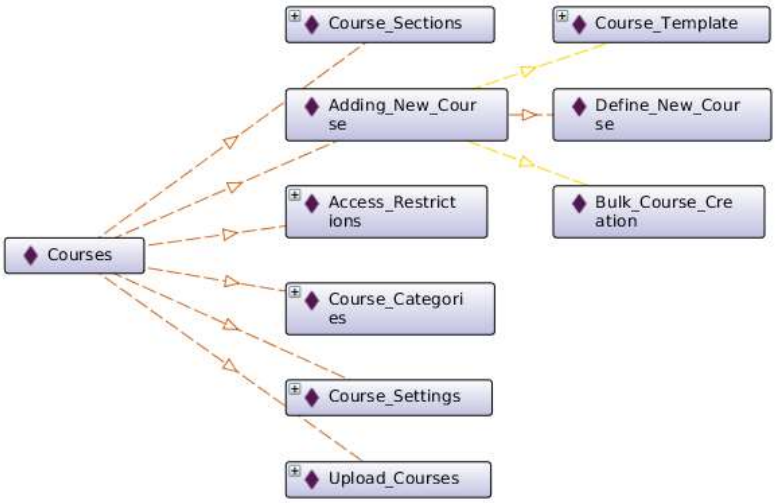

Figure 91. A Graph Representation of the Features' Instances

7.4.3 SPARQL Queries for Extracting Information

After enhancements or bug fixes that are reported by end-users during the maintenance phase, the feature specification is likely to be updated. It was one of the main purposes of the PD study to design with the engineers the structure of the feature specification documentation, and design the guidelines for extracting the necessary information that will be used in the update process. As discussed with them that the effort spent in the documentation could be waived by an automated process that was designed with them and explained in section 6.3.3. This process utilizes the structured feedback thread that took place during the communication between end-users and engineers to extract the necessary information needed to update the feature specification. In this section the queries designed to extract the needed information to fill up the feature specification with updates are explained.

The queries conform to the design of the allowed feedback types and levels of detail to fill the FSD sections, which was set with the participants in the PD study, and summarized in table 19. Figure 92, shows the implemented query used to extract the information for the Description Section. Description Sections can be filled with any type of level of detail except concise or scenario. The feedback thread instances that were queried contained a Topic Definition, 2 Feedback Elaborations, and 2 Investigation Elaborations. All their content was extracted to be added in the description section by the engineer as shown in Figure 93.

For the Investigation elaborations in the query results on rows 4 and 5, the question asked by the engineer was also added beside them, which was shown separately in Figure 94 (due to spacing reasons). Questions should not be included in the description section. However, this will help the engineer re-phrase the requirements in a more meaningful way. For example, the Investigation elaboration row appears as *"I'd rather have it available all the*

time”. However, by displaying the question for the engineer, he will be able to rephrase it to *“Clone setting option will be enabled and available to the users all the time, and not only the first time have they created the course”*

Snap SPARQL Query

```
PREFIX fb: <http://www.aast-bu.edu/fb#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT ?y ?x ?c ?qmc
WHERE {

        {{?x  rdf:type fb:LevelOfDetail}  {?x fb:details ?y}}
        {{?y rdf:type fb:Topic} UNION {?y rdf:type fb:Addition} UNION {?y rdf:type fb:Elaboration} UNION {?y rdf:type fb:ProblemCorrection}}
        {?x fb:describedBy ?z }
        {?z fb:content ?c}
        OPTIONAL {
                {?x fb:linkedTo?q}
                {?q fb:describedBy ?qm}
                {?qm fb:content ?qmc}
        }
        MINUS  {
                {?x  rdf:type fb:Scenario} UNION {?x  rdf:type fb:Concise} UNION {?x  rdf:type fb:ScenarioStep}
        }

}
ORDER BY DESC (?y)
```

Figure 92. A SPARQL Query for Extracting Description Section Items

?c
There is a course that I have defined its format earlier. I want to clone these settings to a newly created course.@en
The absence of this feature is very time consuming and practically exhaustive.@en
Screenshot of the settings menu that allows the user to copy content to a new course.@en
I's rather have it available all the time.@en
They do not have to be from the same course.@en
I am concerned with the accessibility of this feature. We need the staff members to be able to clone from their own courses only...
There are multiple courses that have predefined settings. Professors don't usually inherit settings from other professors, but ra...

Figure 93. The Query Results for the Description Section Items

Will the "Clone Settings" option be valid the first time you create the new course only? Or should it be enabled all the time?@en
Are the cloned content and settings extracted from the same course?@en

Figure 94. The Investigations of the Elaborations available in the Query Results

Figure 95 shows the implemented query used to extract the information for the Scenario Section. Scenario Sections can be filled with scenario levels of detail only, where each scenario contains several scenario steps. The feedback thread instances that were queried contained a Proposal feedback type that was detailed by a scenario level of detail that was explained in 4 scenario steps. All their content was extracted to be added in the scenario section by the engineer as shown in Figure 96.

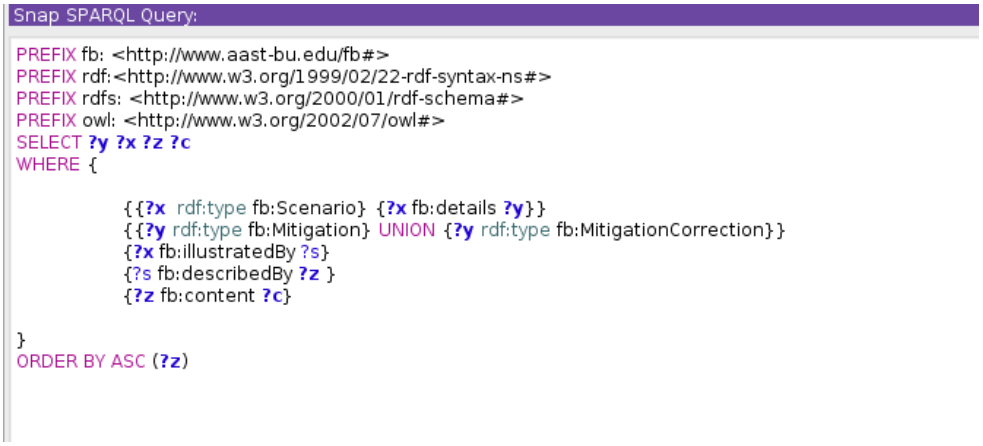

Figure 95. A SPARQL Query for extracting Scenario Section Items

?c
After cloning the existing course an existing course, open the course details page.@en
Click on the new button "Copy from template course".@en
A popup will be shown to search for an existing course.@en
Select the required course then click OK.@en

Figure 96. The Query Results for the Scenario Section Items

Finally Figure 97 shows the implemented query used to extract the information for the Feature relation section. Feature relation Sections can be filled with features deduced from feature links used in the scenario steps that are included in the scenario item. The feedback thread instances that were queried contained a Proposal feedback type that was detailed by a scenario level of detail that was explained in 4 scenario steps. Step 1 used an existing implemented feature, while step 2 used a new feature that was proposed as part of the solution. The feature names were extracted to be added in the feature relation section by the engineer as shown in Figure 98. Also, by the query results the engineer could update the feature model by adding the new proposed feature with its relation.

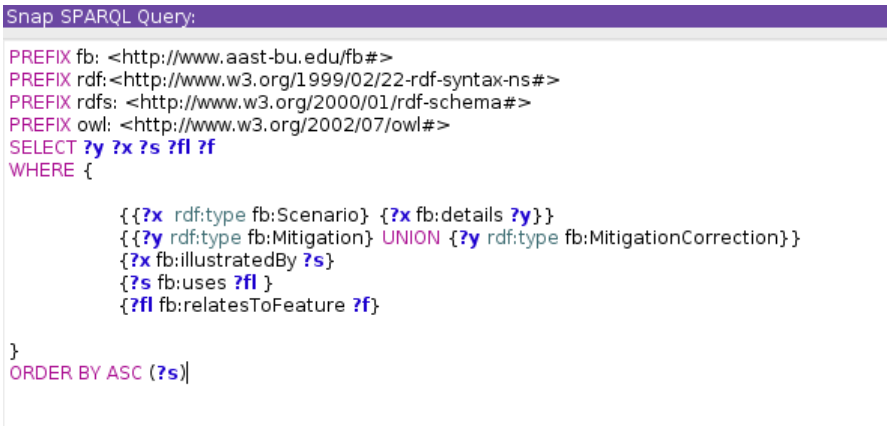

Figure 97. A SPARQL Query for Extracting the Feature Relation Section Items

?f
fb:Using_an_Existing_Course_as_a_Template
fb:Course_Settings_Cloning_Course_Settings_Along_with_course_details

Figure 98. The Query Results for the Feature Relation Section Items

Figure 99 represents a sample filled in feature specification. The sections filled are: Description, Scenario, and Feature Relation. The query results could be directly used as the section's item or could be edited to be more useful or meaningful. For example in the FSD below the engineer edited the Investigation Elaboration to be more meaningful than the direct answers that were provided by the query.

Using an Existing Course as a Template	
Priority:	Essential | Expected | Desired | Optional
Effort:	Months | Weeks | Days | Hours
Risk:	Dangerous | 3-Risks | 2-Risks | 1-Risk | Safe
Functional Area(s):	Word, Word, Word
Use Case(s):	UC-01
Description Section:	?c There is a course that I have defined its format earlier. I want to clone these settings to a newly created course.@en The absence of this feature is very time consuming and practically exhaustive.@en Screenshot of the settings menu that allows the user to copy content to a new course.@en *The feature will be enabled and available to the users all the time, and not only the first time had they created the course* *The cloned content and settings do not have to be from the same course* I am concerned with the accessibility of this feature. We need the staff members to be able to clone from their own courses There are multiple courses that have predefined settings. Professors don't usually inherit settings from other professors, bu
Scenario Section:	?c After cloning the existing course an existing course, open the course details page.@en Click on the new button "Copy from template course".@en A popup will be shown to search for an existing course.@en Select the required course then click OK.@en
Feature Relation Section:	?f fb:Using_an_Existing_Course_as_a_Template fb:Course_Settings_Cloning_Course_Settings_Along_with_course_details

Figure 99. A Sample FSD Filled with the Feedback Instances Obtained from the Query Results (amended from Robbins 2004).

7.5 Ontology Validation

One of the key stages of the ontology authoring process is the implementation of the appropriate specifications to guarantee that the ontology structure and design adheres to a set of common best practices. Reasoners are useful for automatic checking of the logical consistency of ontologies, by performing operations such as equivalence and instantiation checking, but assume that the ontology has been properly structured in the first place in order to produce valid results.

During the ontology authoring process, a set of conditions should be applied to assure process validity. These conditions are used to check the degree of completeness of

achievement of the following criteria: concept and property hierarchy, documentation and visualization, definition of ranges for property values, disjointness restrictions and adherence to naming conventions.

7.5.1 Internal validity (validity of relations/structure)

This type addresses the quality of the inner structure of a model or a theory (Kehagias, Papadimitriou et al. 2008). For example, how accurate are the interrelationships between variables? One way to ensure internal validity of the ontology is the use of reasoners. In this research HermiT (Shearer, Motik et al. 2008), which is a reasoner for ontologies written using the Web Ontology Language (OWL) was used. HermiT provides a command-line interface for common reasoning tasks, including classification and query answering. HermiT is a reasoner that fully supports the OWL 2 standard: it supports all of the datatypes specified in the standard, and it correctly reasons about properties as well as about classes. Although not always the fastest, HermiT exhibits relatively robust performance in the ontology testing, and as shown in the results, it never failed to classify the feedback type instances, or to extract the complete information needed to create the feature specification.

Internal validation takes part as a parallel task performed along with the structuring and enhancements that are made to the ontology. In this research, the feedback types and feature specification, along with selected levels of detail and methods were implemented and the reasoner could infer them properly. Object properties were added to add further controlling on the usage of feedback types, creating and maintaining the feature models, and in providing more accurate query result to create updated versions of feature specifications.

A validation case was performed for the ontology that was described in details in section 7.4. A sample feedback thread that took place during one of the PD design sessions was used to create instances of structured feedback. This is because there are no data sets that could be used for such purpose, as this is a novel structuring method that hasn't been used yet. Also, it was one of the PD study purposes to practically test whether the classification and rules are useful enough for both end-users and engineers to provide meaningful feedback without hindering their experience, which was positively appraised during the study. The feedback components and content conformed accurately to the ontology rules and thus were successfully validated and stored in the ontology knowledge base as explained in section 7.4.1.

Furthermore, instances of feature models were created. These models were also developed and used in the PD sessions by the engineers for impact analysis tasks, and to study how they could be maintained along the evolution process. The instances were created using the feature class and detailed object property list developed to link the features together

according to the feature model notation. This also validated the use of the object properties developed for linking purposes and helped in building visual feature models on the ontology.

Finally, the essential step for validating the ontology structure and usage was to show that the stored instances in the ontology knowledge base are sufficient to produce adequate query results that could be used to maintain the feature specification structure, which was explained and shown in details in section 7.4.3.

7.5.2 External validity (scope of applicability)

External validity is related to generalization (El-Diraby 2014). How general is the model? What limitations are applicable to the model? Under which conditions, can the proposed model be applied reliably. Typically, statistical methods are used to test the applicability of a model to its claimed domains. Several techniques are used to ensure external validity of the developed ontologies. Academic peer review and interviews are common methods for confirming external validity.

In our research, building the ontology is not itself a goal. Instead it acts as the rule engine component, which is part of accomplishing a higher level aim of designing an acquisition method for utilizing structured user feedbacks that could be used to keep requirements' information up-to-date during maintenance phase. One way to formalize the guidelines that govern the feedback types, feature models and feature specifications was defining them using ontology. This adds further advantages to how well it handles the extendibility of its domain and scope, and it becomes clearer if there are adjustments needed in ontology and what are they. Therefore, the need to separately validate the ontology was waived, because mainly it is being used in a very specific and narrow domain. Still, performing validations using reasoners and competency questions was made to ensure the scope of the ontology was met.

However, at the final stage of this research the PD study, conducted and explained in chapter 6, reported how well the acquisition method handles the structuring and storage of user feedbacks and designed the guidelines for utilizing them in requirements' documentation. Thus, providing an ontology design that strictly adheres to the study results and design rubrics provides a validation on how well the ontology performs its intended tasks.

7.6 A Tool Mock-up for Updating Requirements Information

In this section a tool mock-up for aiding engineers in updating the feature models and the feature specification structure is demonstrated and explained.

This tool mock-up was developed to help in visualizing how a real software system that utilize the developed method would look like, and move it from just concepts to reality where it could be benefited from.

The final phase in the feedback acquisition and communication method is the sub process for updating feature model and feature specification with information extracted from the communication thread. This is the ultimate utilization of feedback in this thesis work. This mock-up uses the same example feedback thread that was used as a feedback instance in the ontology.

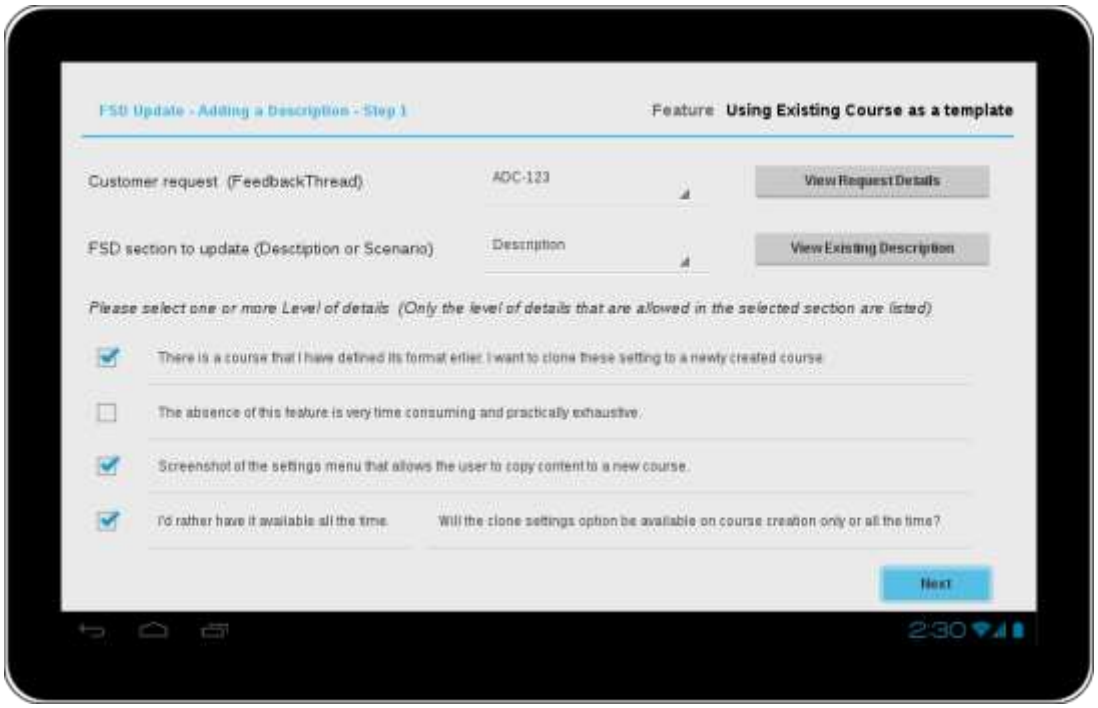

Figure 100. A Mock-Up Screen for Extracting the Feature Specification's Description Section

The first screen in Figure 100 above is a mock-up screen designed to help engineers extract the needed information to update the description section. On the header of the screen appear the step name, which is "Adding a description-Step 1", and the topic feature of the feedback thread.

The engineer then specifies the specific customer request (i.e. feedback thread) from which he wants to extract information from. The button beside the feedback thread is designed to enable the engineer to view the whole feedback thread when needed.

Next the engineer specifies which section in the feature specification he intends to update. In this screen he intends to update the description section. The button beside the description section allows the engineer to view the existing description (if any). This is to decide whether the new information he intends to extract are contradicting to what exists, overrides them, or adds to them.

Then a list of all eligible levels of detail that could be used to update the description section that occurred in the selected feedback thread are displayed. This is to enable the engineer to select one or more items to form the new description.

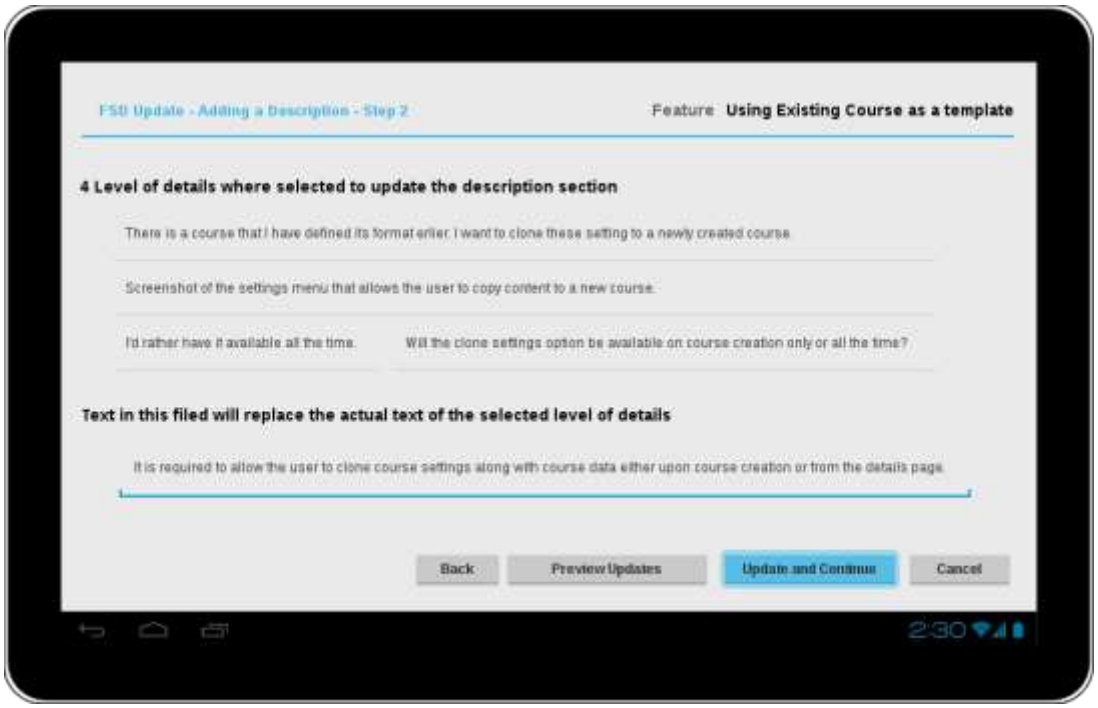

Figure 101. A Mock-Up Screen for Updating and Previewing the Feature Specification's Description Section

When the engineer clicks on the bottom "Next" button in the screen in Figure 100, the system navigates to the next step in the description addition process, which is the actual updating and/or previewing the updates, this next step in demonstrated above in Figure 101.

In that Figure, on the header of the screen appear the step name, which is "Adding a description-Step 2", and the topic feature of the feedback thread. Next, the selected levels of detail from the previous step are displayed. Then, the engineer is given the capability to update them as they are or edit and summarize them (while preserving the link to the actual thread and selected levels of detail).

The buttons in the bottom of the screen enables the engineer to go the previous step to re-select the needed items, to preview the final updates for the description section, to perform the actual updating in the specification and proceed to the next step, or to cancel.

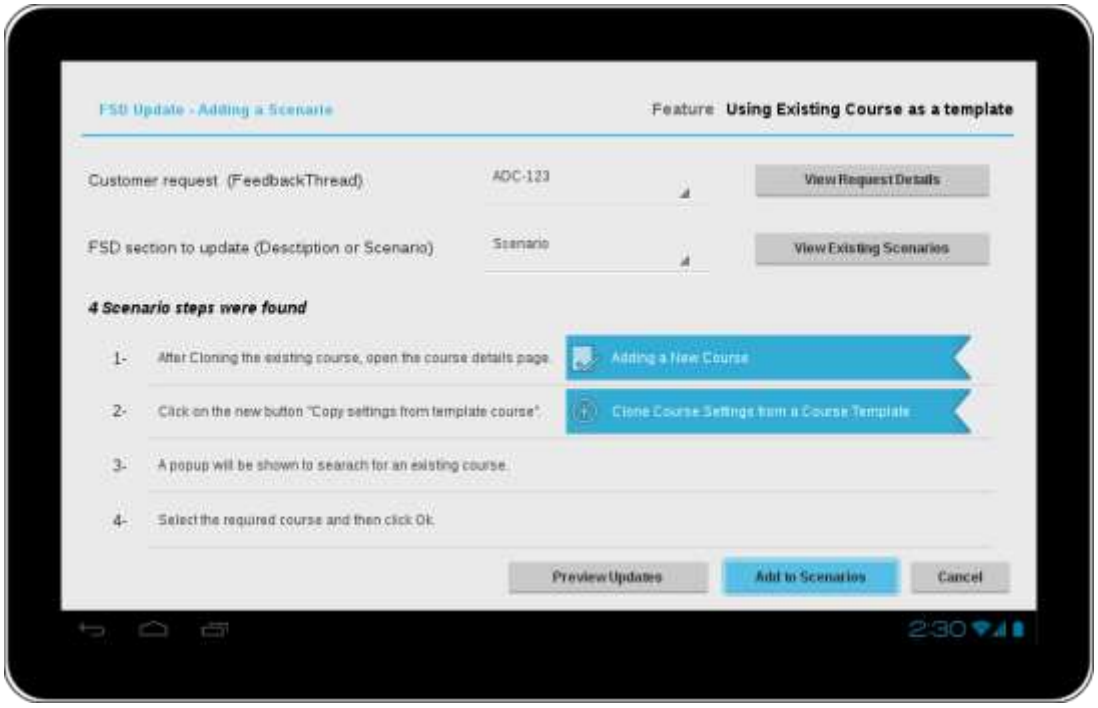

Figure 102. A Mock-Up Screen for Extracting the Feature Specification's Scenario Section

Figure 102 above shows a mock-up screen designed to help engineers extract the needed information to update the scenario section in the feature specification. On the header of the screen appear the step name, which is "Adding a Scenario", and the topic feature of the feedback thread.

Same way as in the description section the engineer is required to specify the feedback thread he wishes to extract from and selects that he wants to update the scenario section.

Automatically, the tool is designed to extract the scenario from the solution in the thread with all the steps listed as shown. If a step in the scenario utilizes certain features then they are displayed beside the step they relate to as shown in Figure 101 in steps 1 and 2.

In the bottom list of buttons the engineer can either choose to preview the scenario before updating the section, directly updating the scenario section, or cancel.

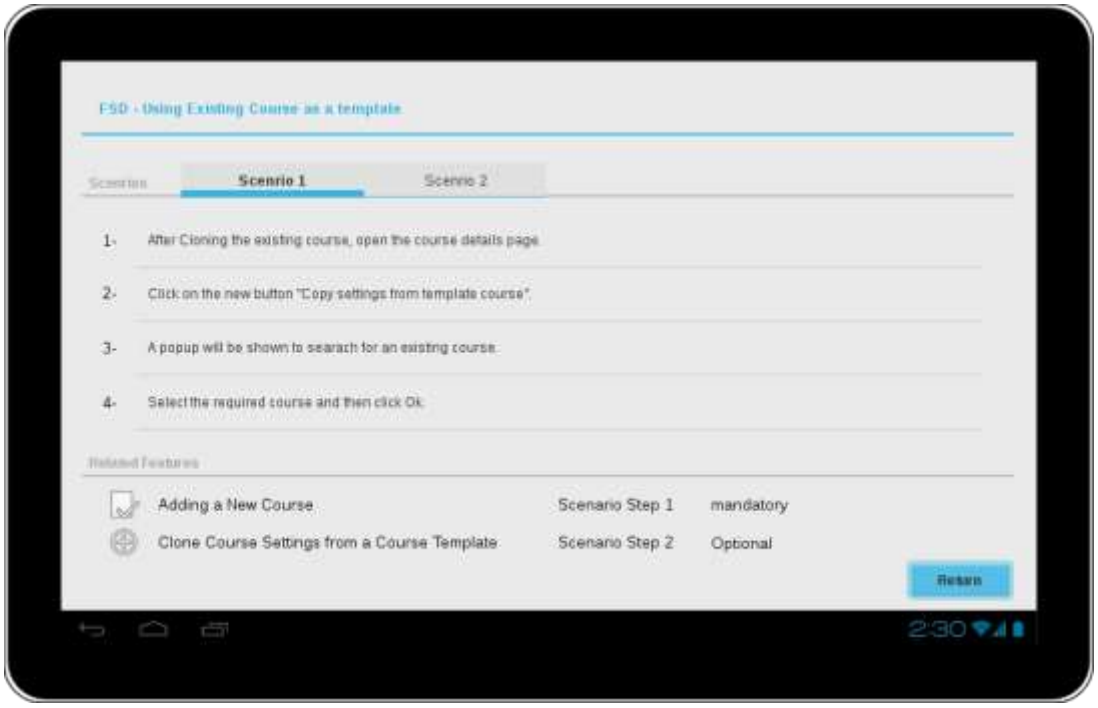

Figure 103. A Mock-Up Screen for Previewing the Feature Specification's Scenario and Related Features Section

Finally, Figure 103 shows the outcome if the engineer choses to preview the feature specification scenario section, the available scenarios will be previewed in tabs with the latest addition in the first tab. Several scenarios may exist for a single feature, as a feature can be used in several contexts differently.

Furthermore, the list of utilized features in the selected scenario occurs in the related feature section. In this list the feature name along with the step number in the scenario it is used in, and the type of relationship that connects it to the main topic feature of the thread.

Finally, the icons on the left of the feature names specify the type of feature. For example, the first feature used in step 1 is an existing feature in the feature model with a mandatory parent child relation with the topic feature. In step 2 the plus icon is used to indicate that this is a new proposed feature which should be added to the feature model with an optional parent child relation with the topic feature. This update could be managed automatically, since all the necessary information is encapsulated in this step.

7.7 A Framework for Runtime Communication and Requirements Updating

In this section a framework that illustrates how the processes, methods, and models developed in this research are employed to achieve the research goals is explained and illustrated in figure 104.

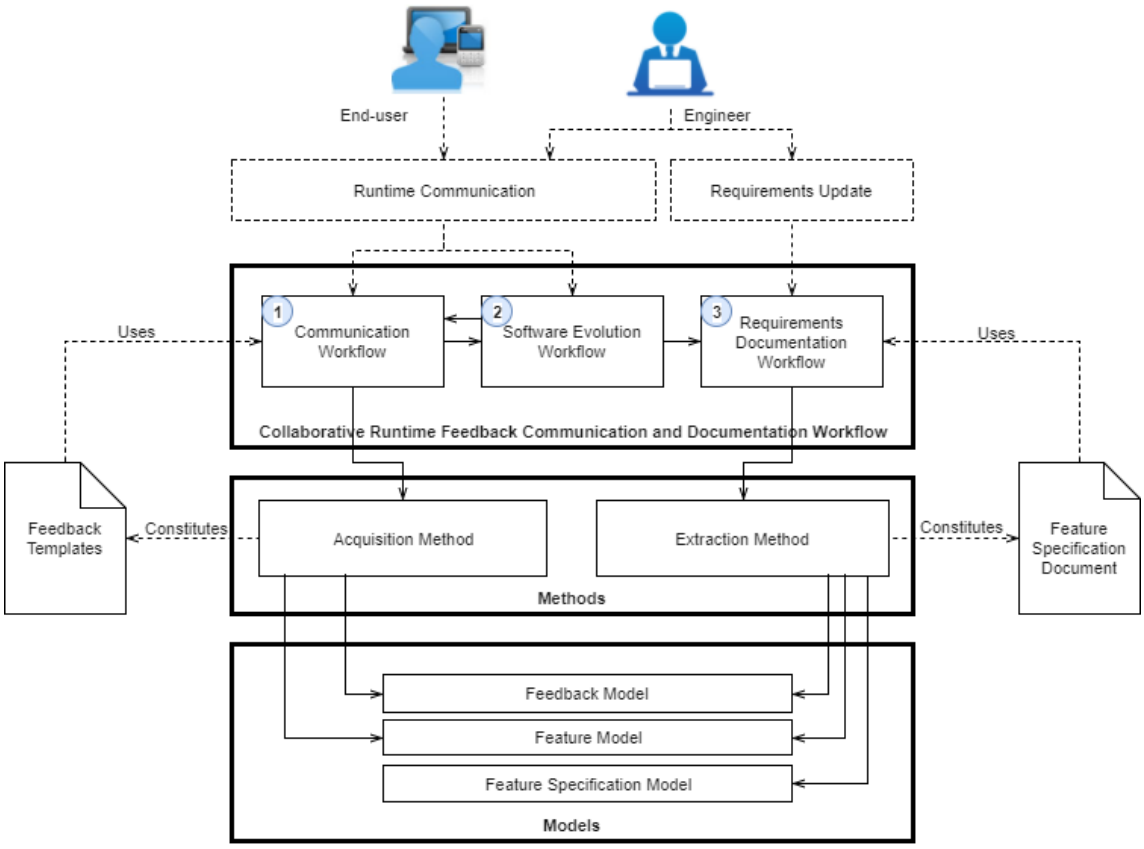

Figure 104. A Sociotechnical Framework for Runtime Communication and Requirements Updating

The framework is explained in the following list:

Roles Involved:

- **End-user:** is involved in this framework to provide feedback about the system's quality or change requests.
- **Engineer:** resolves any software quality issues or change requests. Also, the engineer utilizes the information gathered in this communication in updating requirements.

Main Tasks: summarize the main goals that this thesis work focuses on.

- **Runtime Communication:** the main task that both end-users and engineers collaborate together in is the runtime communication task. Both roles communicate through feedback to form feedback threads of communication that could be further utilized.
- **Requirements Updating:** One main task that engineers have to perform by the end of their communication with end-users is keep requirements information up-to-date.

Workflows: These workflows are the process that was developed in this research shown in Figure 40 in section 6.3.1.2. This process integrates three main workflows together to achieve the main tasks, which are:

1) **Communication Workflow:** these are the steps in the process concerned with the interaction between end-users and engineers through obtaining their feedback. From the process in Figure 39 examples are: *"Add a New Problem (Topic Definition)"*, and *"Add New Information to the Problem (feedback Elaboration)".*
2) **Evolution Workflow:** these are the steps in the process concerned with the change identification and evolution tasks. These are well-known tasks were adopted from the existing processes in the literature as explained in section 2.7.1. From the process in Figure 40 examples are: *"Identify the Problem"* and *"Analyse the Problem".*
3) **Requirements Documentation Workflow:** this is the final step in the developed process that is responsible for documenting the updates that occurred along the communication thread. From the process in Figure 40 the sub process name is: *"Update the feature model and feature specification document".* It was modelled as a sub process because in order to achieve it, there are other internal processes and decisions that have to be made. Figure 45 in section 6.3.3 shows the internal flow of the update sub process.

The tasks are numbered in chronological order, where the communication initiates the collaboration at runtime between the engineers and the end-users. This is followed by the engineers performing the evolution process tasks. This is repeated until the evolution process ends with a confirmation from the end-user. After that the requirements documentation takes place.

Methods: this summarizes the tools that were developed and utilized in this research for the above name workflows:

- **Acquisition method:** as mentioned in the communication workflow that the interaction between the two roles is achieved through obtaining their feedback. Feedback acquisition is a method that employs the set of feedback types that were developed at earlier stages of this research. Each feedback type has a meaning that serves a certain purpose in the communication. Also, each feedback type has a set of constituents that ensure that minimum useful information is provided. More elements can be used to provide extra information if needed.

  The acquisition method uses the feedback and feature models structure and rules defined in the ontology to constitute the *feedback templates* that are used during the communication workflow to acquire structured feedback.

- **Extraction method:** In order to support the requirements documentation workflow a set of rules for extracting information were developed to ensure that adequate levels of detail are extracted from the qualified list of feedback types as summarized in table 19 in section 6.3.3. Accordingly, a set of queries were developed for extracting these levels of detail for each section (description, scenario, and related features) of the novel structure of the *feature specification document*. Also, the extracted information for the related feature section is used to update the feature model by adding features and/or relationships.

Models: Ontology design was used to build the entities (classes of concepts), relationships between them (object properties), and the rules that govern their usage. The ontology rules helped in validating the feedback instances and facilitated the queries' construction.

- **Feedback Model:** defines the classification of feedback types, their levels of details, and the methods used to describe them.
- **Feature Model:** a basic representation of a feature (implemented or proposed) to differentiate between existing features, and new ones introduced in change requests. Also a set of object properties, extracted from existing literature of feature model notation, were defined to relate features together to constitute feature models.
- **Feature Specification:** the novel representation of feature specification structure that defines the description, scenario, and related features sections and the items used to fill them.

7.8 Summary

In this chapter three model designs were implemented using ontology structures and rule. The three models are: the feedback model that represented the final classification of feedback types and their constituents, the feature model for representing the existing software features, and the feature specification model that represents the new structure for documenting features. Furthermore, a case study was introduced to validate the implementation and show its usage. A sample thread of structured feedback was used as an instance, validated and stored in the ontology knowledgebase. In addition to that a sample feature model for existing features of the entered feedback thread was constructed and stored in the ontology. Both were utilized and queried using SPARQL to obtain the necessary information to update both the feature model and feature specification documentation. To further visualize how an application for engineers could help realize the developed concepts, mock-ups were designed to provide a walkthrough the process.

In the next chapter the contributions of this research are explained and the conclusions showing the benefits of this work to practitioners and summarizing the research outcomes in

the form of a framework for runtime communication and requirements updating. Finally, possible extensions to this research are explained.

8. Contributions, Conclusions, and Future Work

To conclude this thesis, this chapter presents the thesis contributions to knowledge, concludes the thesis, and discusses possible future work.

8.1 Contributions of this Research

This thesis contributes to knowledge as discussed in this section.

Contribution 1 – Identification of the key concepts and challenges for acquiring crowdsourced software evaluation:

In the first study of this research the potential of crowdsourcing for software evaluation was advocated. This was the first exploratory study in this research and the aim was to examine with actual users their needs and expectations on the activity of acquiring evaluation feedback from the crowd. The collective end-users' judgements can enrich and keep the timeliness of the developers' knowledge about software evaluation via their iterative feedback. Although this seems promising, crowdsourcing evaluation introduces a new range of challenges mainly on how to organize the crowd and provide the right platforms to obtain and process their input.

This work introduced a set of concepts needed for the correct design of acquisition methods of evaluation feedback. At the same time, the identified concepts (features) make its correct implementation challenging. Thus, among the various challenges, those informed by the focus group results and related to obtaining users evaluation feedback were identified and discussed. This helped in enriching the area of crowdsourcing evaluation by discovering new dimensions that could help move the field forward and provided design elements for developing new approaches and platforms for acquiring crowdsourced feedback.

Contribution 2 – Defining the feedback concept, its constituents and the rules that govern their usage in the business software maintenance domain:

It was concluded from the existing literature, that the definition of feedback lacks the characterization of its details and behaviour. This hinders the ability of researchers in utilizing it to develop more formal and systematic techniques for feedback acquisition, which in turn affects many areas and disciplines such as requirements analysis and extraction, requirements documentation, software evaluation, maintenance and evolution, and many others.

This research contributed to the body of knowledge by defining: 1) what is a feedback, 2) developing a classification of different feedback types that could be used by both end-users and engineers in their communication during maintenance, 3) defining the set of elements

that constitute the mandatory information for each feedback type and that can also be utilized to further enrich the definition of any feedback to ensure it provides useful and meaningful information; 4) finally defining the set of rules that govern the feedback usage and linking to other feedback types to form threads of communication.

Contribution 3 – Identifying the role of feedback in accomplishing evolution tasks, and identifying the engineers' needs and challenges

In this research a study was designed to capture the engineers' perspective of the role of feedback in accomplishing evolution tasks. This study captured a group of concepts regarding the missing information that directly affects the evolution task accomplishment and decision making, and another group of concepts was captured regarding the problems that result from these types of missing information and miscommunication in the maintenance phase.

In this research this study helped deriving the rest of the research outcomes. It confirmed the effectiveness of the developed feedback structures from the engineers' perspective. It highlighted the need for a communication means aligned with the main tasks the engineers perform in the evolution process. It stressed on the essence and need for updated requirements information, and captured its effect on all the change management and evolution tasks.

More results could be obtained from this study that could help develop an enhanced framework for utilizing feedback. For example, thematic associations could be captured from analysing the relations between the sets of concepts: problems and missing information, to form patterns of recurring situations. These situations could guide engineers in the maintenance phase by providing historical information and lessons that could improve their decisions (discussed in section 8.3).

Contribution 4 – Developing a new formal systematic method for feedback acquisition and communication

Based on the classification of feedback types, its constituents and rules that were developed early in this research, a formal and systematic feedback acquisition method was developed. This method is formalized through the use of ontology for feedback structuring, validation and storage. Furthermore, it provides systematic means to capture and communicate the feedback while performing the normal evolution tasks. Aligning this new process to the well-established evolution process adds on to its usefulness as it helps inform each task in the evolution process which leads engineers to produce accurate results, and help users take more active role in the process.

Moreover, this process employs the use of RE models specifically feature models, which has several benefits: 1) enables the automatic retrieving of feature naming, types and relationships to inform tasks such as impact analysis, 2) linking the acquired feedback to features helps engineers to correctly identify the modification scope, and help keep requirements information updated depending on the type of change communicated in the thread.

Contribution 5 – Developing a new systematic method for extracting requirements information for updating the feature model and specification:

The novel feedback acquisition and communication method that was developed in this research also ensures that when engineers provide their solution it is well linked to features in the feature model, and specifying whether the features employed in their solution already exist or newly proposed. This ensures that RE models' updates are captured during the communication process.

Furthermore, the final phase of the communication process "Update the feature model and feature specification" reassures the importance of keeping the requirements up-to-date for future changes. A detailed systematic process was developed to guide engineers through the requirements extraction and updating. This process gathers the information from the feedback thread acquired during the communication, enables the engineer to filter or summarize it, and automatically updates the feature specification document.

Moreover, a new structure was developed for feature specification to ensure minimum amount of useful information that could benefit the end-users in understanding and using the system, and at the same type help engineers in their identification and analysis of future changes.

Contribution 6 – Implementing a formal method for validating, storing, and utilizing feedback and creating tool prototypes:

The ontology design developed in this research provided explicit formal specifications of the concepts in the domain of feedback definition, its utilization, and relations among them. This contributed to the research community and software engineers by sharing a common understanding of this novel feedback classification. This will enable its reuse, modification, and extendibility to include further related concepts and refinements to enrich its usage.

Furthermore, two types of mock-ups were developed in this research. The first one was for obtaining feedback and utilizing the feedback elements in the acquisition process to provide useful and meaningful information. This mock-up tool also showed how feedbacks can be linked to the business process and features of the system. Toolboxes were also designed to guide the users and help them in the task. By creating a mock-up, it was possible to sit

down with an imitation of a real version of the product and determine which aspects are worthwhile and which parts need to be revised or discarded. In this process, it was possible to find noticeable gaps that, on paper, weren't noticeable. Additionally, creating a mock-up allowed the research team to evaluate the idea and test its actual usage.

The second mock-up was designed to further benefit from the ontology implementation results, which entail the utilization of stored feedback instances to update feature models and the new structure of feature specification. Just like it is much easier to see if there are problems with a design by having a mock-up, it also helps to present new ideas or concepts to potential researchers' community when they have a mock-up to visualize its interface and usage. Without mock-ups, the research outcomes are only concepts, which can make it difficult to get potential researchers or even the industry to commit to the adoption of the results.

Contribution 7 – Developing a sociotechnical framework for runtime communication and requirements updating:

A final contribution to the literature is reinforcing the research outcomes with a sociotechnical framework for runtime communication and requirements updating illustrated in Figure 104. This engineering framework provides a standard way for both end-users and engineers to communicate using feedback and benefit from this communication in requirements documentation updating. This framework shows the basic components and the relationships between them. It incorporates end-users and engineers, their goals, the new developed workflows, the methods, and the ontology models used to store, retrieve, and reuse information. Also, it provides a view of their order, and the dependencies between them. Finally, it shows the new artefacts that were developed in this thesis, which are the feedback templates and the feature specification documentation, their constitution and usage.

8.2 Conclusion and Benefits

This research was motivated by the need for developing a new formal systematic acquisition and communication method due to the lack of such an approach. Having a formal definition for feedback structures, their constituents, and the rules that govern their use positively improve the feedback quality, which consequently affects the communication between end-users and engineers and the success of the software evolution. Also, the systematic way for acquiring and communicating feedback enabled the development of systematic means for keeping requirements information up-to-date during the maintenance phase. To reach the aim of this thesis, several studies and software engineering processes and paradigms were conducted/ used to collectively help reaching this aim, and answer the research questions.

There are several benefits to the concepts adopted in this research that is:

- **Evaluation in real context**, i.e. evaluating software when users are using it in practice and out of labs.
- **Access to a wider and diverse set of users and contexts of use** that were unpredictable by analysts. This approach allows users to act as the actual validators of the system and give feedback regarding each alternative, enhancing the overall quality of the system functionality and behaviour. For example, in using cloud computing services it cannot be certain how users belonging to different organizations, cultures, expertise, etc. will perceive the software and there is no way to predict all that variety in the sample of users that would be used in a traditional usability study.
- **Validating highly-variable software**, potentially with reduced cost and minimized time. Iteratively obtaining and processing feedback to support evaluation helps to accelerate the evolution process. This is particularly true for highly variable systems with a large number of alternatives. Such validation was done by the designer at the usability testing, and user centred design. A large number of users were required to evaluate the entire alternatives. This approach was very expensive, time consuming, and hardly manageable. Things become even worse when the user business and IT worlds change so that a re-validation has to be done.
- **Informs the software evolution process**, e.g. by introducing a more formalized structure for feedback that enables its systematic usage to provide useful and meaningful information. Augmenting the evolution process with a communication workflow to systematically collect information needed to accurately accomplish the evolution tasks.
- **Maintaining the requirements information up-to-date.** That is by utilizing feedback threads to extract new/updated requirements that could accurately help both end-users and engineers each on his side. Thus, evolving the system in a more systematic and accurate manner to better meet the users' needs and expectations.

8.3 Future Work

This thesis's future work is discussed in this section.

1) Additional RE models utilization

In the participatory design study that was conducted, the use of Business process models was introduced to the participants. The purpose was to try to link the feedback to features and to narrow down the feature selection according to the business process the end-user was working on when the problem occurred. However, this was practically waived in the design sessions by all participants, as the used fictional scenarios were all designed to result in technical enhancements for a specific feature. As a future work, the benefits of this

model could be further evaluated by designing scenarios that should result in process enhancement, which cross-cut different business modules.

Also, in this research feature models were used to link feedback to a certain feature. However, a further level of detail could be added by employing goals models as well. Goal models could help engineers in prioritizing the problems according to goals; also it could help the managers in marketing new change requests based upon their knowledge with the customer's goals, and evolution patterns.

2) Extracting and modelling problem patterns

The interviews' study with engineers that was conducted in this research could be further analysed to extract additional information that could help expand and enrich the research outcomes. The purpose of the analysis will be to extract most frequent problems that engineers encounter and relate them to the type(s) of missing information that causes them to form pattern of frequent situations.

A Problem Register model could be developed. This register will contain details about the frequent problems that engineers encounter during the change management process. The interviews study that was conducted indeed does not capture every single problem that occurs during the software maintenance; Therefore, engineers will be given the capability to enter new problems and categorize them in the problem register under the suitable phase they were working on (i.e. problems are categorized by the phase they occur in). This is to provide flexibility and extensibility to the model.

During the investigations the engineers will be asked to link the type of missing information they are investigating to specific problem that resulted due to lack of this information. Both the types of missing information and the problems will be stored in the problem register. This will form a pattern of linked problems that were encountered during a certain phase in the change management process that can be used in the future to prevent similar cases and increase engineers' awareness of potential risks.

3) Extracting and developing history models

A History model carries information about both **feedback history** and **feature change history**. Both types of historical information can be derived from the formal threads of feedback types stored in the ontology knowledge base.

An issue tracking systems could be integrated with the **feedback history model** to further enhance its capabilities. An **Issue log** is a documentation element that contains a list of ongoing and closed issues of the project that is stored through issue tracking systems. For each issue the list of information that was gathered by engineers at different phases of software change process, which led to better understanding of the issue on hand will be

also retrieved and recorded. This information combined together can feed the recommender system with valuable inputs.

Finally, the **feature change log** is a subset of the feedback information history but with the focus on feature changes requested by customers. Customers may report invalid problems, help requests, bug fixes, enhancements or new features. It is important to keep complete documentations of feature changes or new features in order to update the requirements documentation for future usage. The feature change log carries the combined information to serve that purpose.

4) Recommender Systems Utilization

From the interviews' study that was conducted, information was gathered regarding evidence about the types of missing information that hinders the engineers' capabilities in accurately performing their tasks, and their ability to deliver proper outputs in a timely manner. This led to lots of misused efforts in performing repeated tasks and investigations. In this research it has been argued that recommender systems can play an important role in reusing the existing information in reducing the human interventions and effort. Example of cases the can aid both customers and engineers are provided below and will be explored in the next participatory study.

Solution Reusability: When the user enters a problem, the problem could compare with other stored problems in the knowledge base using: Feedback Type, Specified Feature, Business Process, Activity in the Business Process, Problematic Step. The knowledge base is analysed and relevant cases are retrieved and displayed to the user using the analysis and query method in the framework. If the user identifies a similar problem and it has a verified solution then the system helped him in resolving the issue.

Problem's Prioritization: If the user identified a similar problem with no solution (i.e. an ongoing issue) then he could be asked to vote that he has a similar problem to increase the problem's priority. If the customer did not vote then the retrieved cases with the new problem could be then displayed to the software engineer whom by turn will vote it as similar to increase its priority. Also, if the user found a similar problem with its solution and did not try it then the developer can return it as a duplicate and suggests that the user tries it.

Missing Information Investigation: When a new problem is entered by the customer, the analysis and query tools could be utilized to start searching the History model (discussed in the previous point) to obtain previously entered relevant cases that were investigated and successfully resolved. After retrieving the relevant cases containing: the issue, information gathered when investigating that issue, and the resolution status, the recommender system starts to detect similarity between the new entered problem, and the issues in the History model. When similarity is detected the recommender system starts advising the software

engineer with the types of missing information that he should collect and he decides whether they are relevant to the new problem or not. This increases the engineer's awareness of problems that may occur.

Risk Assessment: Also once the engineer starts the change identification phase and asked an investigation related to a certain type of missing information and a specific type of problem from the problem register, this triggers a possible problem pattern to be retrieved. Therefore, the engineer and other team members working on the issue could be notified with the potential risks that they may encounter during the analysis, and planning.

5) Integrating with Strategies and Design Principles of Digital Motivation

In this thesis work, end-users play an important role as collaborators within the workflow of the feedback acquisition and communication method. Their input directly affects the accuracy of resolving change requests, and informs the software engineers' evolution tasks and decisions. Thus, it is important to keep the end-users motivated to play their roles and accomplish the tasks allocated to them and enhance their efficiency and experience during their involvement. This could be achieved via software, which incentivizes end-users to perform their tasks more accurately through designing rewarding systems (Shahri, Hosseini et al. 2014). Future research entails examination of aspects and concerns of digital motivation (Shahri, Hosseini et al. 2016), and exploring its integration with the acquisition and communication method to encourage end-users to provide useful and meaningful information, commit to their tasks and to be keen to an active involvement in a timely manner. This could help improve the time taken to resolve issues and change requests, and the customer satisfaction.

9. References

Adikari, S. and C. McDonald (2006). User and Usability Modeling for HCI/HMI: A Research Design. *International Conference on Information and Automation, ICIA 2006*.

Adomavicius, G. and A. Tuzhilin (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering,* **17**(6): 734-749.

Akiki, P., A. Bandara and Y. Yu (2013). Crowdsourcing user interface adaptations for minimizing the bloat in enterprise applications. *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, ACM.

Ali, R., C. Solis, I. Omoronyia, M. Salehie and B. Nuseibeh (2012). Social adaptation: When software gives users a voice. *7th International Conference on Evaluation of Novel Approaches to Software Engineering,* Wroclaw, Poland.

Ali, R., C. Solis, M. Salehie, I. Omoronyia, B. Nuseibeh and W. Maalej (2011). Social sensing: when users become monitors. *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of Software Engineering*. Szeged, Hungary, ACM**:** 476-479.

Almaliki, M. and R. Ali (2016). Persuasive and Culture-Aware Feedback Acquisition. In *International Conference on Persuasive Technology* (pp. 27-38). Springer, Cham.

Almaliki, M., F. Faniyi, R. Bahsoon, K. Phalp and R. Ali (2014). Requirements-Driven Social Adaptation: Expert Survey. *20th International Working Conference on Requirements Engineering: Foundation for Software Quality.* Essen, Germany , Springer.

Almaliki, M., C. Ncube and R. Ali (2014). The design of adaptive acquisition of users feedback: An empirical study. *IEEE Eighth International Conference on Research Challenges in Information Science (RCIS).* Marrakesh, Morocco, IEEE.

Almaliki, M., C. Ncube and R. Ali (2015). Adaptive software-based Feedback Acquisition: A Persona-based design. *IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*. Athens, Greece, IEEE.

Anquetil, N., K. M. Oliveira, A. dos Santos, P. da Silva jr, L. C. de Araujo jr and S. Vieira (2005). A tool to automate re-documentation. *Forum of the CAISE, Conference on Advanced Information Systems Engineering* (CAiSE'05). Porto, Portugal.

Archak, N., A. Ghose and P. G. Ipeirotis (2011). Deriving the pricing power of product features by mining consumer reviews. *Management Science* **57**(8): 1485-1509.

Bacon, D. F., Y. Chen, D. Parkes and M. Rao (2009). A market-based approach to software evolution. *Proceedings of the 24th ACM SIGPLAN conference companion on Object Oriented Programming Systems Languages and Applications*, ACM.

Banerjee, N., D. Chakraborty, K. Dasgupta, S. Mittal, A. Joshi, S. Nagar, A. Rai and S. Madan (2009). User interests in social media sites: an exploration with micro-blogs. *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, ACM.

Barone, D., E. Yu, J. Won, L. Jiang and J. Mylopoulos (2010). Enterprise modeling for business intelligence. *The practice of enterprise modeling*, Springer**:** 31-45.

Batory, D. (2005). Feature models, grammars, and propositional formulas. *Proceedings of 9th International Conference on Software Product Lines.* Rennes, France, Springer.

Bennett, K. H. and V. T. Rajlich (2000). Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 73-87). ACM.

Berg, B. L. (2004). Methods for the social sciences. Pearson Education Inc., United States of America.

Bougie, G., J. Starke, M.-A. Storey and D. M. German (2011). Towards understanding twitter use in software engineering: preliminary findings, ongoing challenges and future questions. *Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering*, ACM.

Boyatzis, R. E. (1998). Transforming qualitative information: Thematic analysis and code development, sage.

Brabham, D. C. (2008). Crowdsourcing as a model for problem solving: An introduction and cases. *Convergence* **14**(1): 75-90.

Braun, V. and V. Clarke (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology* **3**(2): 77-101.

Buskermolen, D. O. and J. Terken (2012). Co-constructing stories: a participatory design technique to elicit in-depth user feedback and suggestions about design concepts. *Proceedings of the 12th Participatory Design Conference: Exploratory Papers, Workshop Descriptions, Industry Cases*-Volume 2, ACM.

Cámara, J., G. Moreno and D. Garlan (2015). Reasoning about human participation in self-adaptive systems. *IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'15)*, IEEE.

Castañeda, V., L. Ballejos, M. L. Caliusco and M. R. Galli (2010). The use of ontologies in requirements engineering. *Global Journal of Researches in Engineering* **10**(6).

Charlotte Magnusson, K. R.-G., Konrad Tollmar, Eileen Deaner (2009). User Study Guidelines, *Hapti Map Consortium Project co-funded by the European Commission within the Seventh Framework Programme*.

Cleland-Huang, J., M. Jarke, L. Liu and K. Lyytinen (2013). Requirements Management–Novel Perspectives and Challenges. *Dagstuhl Reports* **2**(10): 117-152.

Clement, A., B. McPhail, K. L. Smith and J. Ferenbok (2012). Probing, mocking and prototyping: participatory approaches to identity infrastructuring. *Proceedings of the 12th Participatory Design Conference*: Research Papers-Volume 1, ACM.

Crawford, H. K., M. L. Leybourne and A. Arnott (2000). How we Ensured Rigor from a Multi-site, Multi-discipline, Multi-researcher Study. In *Forum Qualitative Sozialforschung/Forum: Qualitative Social Research* (Vol. 1, No. 1).

Creswell, J. W. (2007). Five qualitative approaches to inquiry. Qualitative inquiry and research design: Choosing among five approaches **2**: 53-80.

Creswell, J. W. (2012). Qualitative Inquiry and Research Design: Choosing Among Five Approaches, SAGE Publications.

Creswell, J. W. (2013). Research design: Qualitative, quantitative, and mixed methods approaches, Sage publications.

Dalpiaz, F., R. Snijders, S. Brinkkemper, M. Hosseini, A. Shahri and R. Ali (2017). Engaging the crowd of stakeholders in requirements engineering via gamification. *In Gamification* (pp. 123-135). Springer International Publishing.

Dave, K., S. Lawrence and D. M. Pennock (2003). Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. *Proceedings of the 12th International Conference on World Wide Web*, ACM.

de Souza, S. C. B., N. Anquetil and K. M. de Oliveira (2005). A study of the documentation essential to software maintenance. *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information,* ACM.

Deterding, S., M. Sicart, L. Nacke, K. O'Hara and D. Dixon (2011). Gamification. using game-design elements in non-gaming contexts. *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, ACM.

Dougiamas, M. and P. Taylor (2003). Moodle: Using learning communities to create an open source course management system.

Dybå, T. and T. Dingsøyr (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology* **50**(9–10): 833-859.

Eichhorn, B. R. and O. I. Tukel (2016). A Review of User Involvement in Information System Projects. *Project Management: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications*: 1.

El-Diraby, T. E. (2014). Validating ontologies in informatics systems: approaches and lessons learned for AEC. *Journal of Information Technology in Construction* (ITcon) **19**(28): 474-493.

Ernst, N., A. Borgida, I. J. Jureta and J. Mylopoulos (2014). An overview of requirements evolution. *Evolving Software Systems*, Springer**:** 3-32.

Etezadi-Amoli, J. and A. F. Farhoomand (1996). A structural model of end user computing satisfaction and user performance. *Information Management* **30**(2): 65-73.

eVALUEd Project. (2006). Interview process. Retrieved 15 February 2016, from http://www.evalued.bcu.ac.uk/tutorial/4c.htm.

Fereday, J. and E. Muir-Cochrane (2006). Demonstrating rigor using thematic analysis: A hybrid approach of inductive and deductive coding and theme development. *International Journal of Qualitative Methods* **5**(1): 80-92.

Forward, A. and T. C. Lethbridge (2002). The relevance of software documentation, tools and technologies: a survey. *Proceedings of the 2002 ACM symposium on Document Engineering*, ACM.

Foth, M. and J. Axup (2006). Participatory Design and Action Reseach: Identical Twins or Synergetic Pair? *Proceedings of the Participatory Design Conference*.

Fuchs, N. E., K. Kaljurand and G. Schneider (2006). Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces. In *FLAIRS Conference* (Vol. 12, pp. 664-669).

Galvis Carreño, L. V. and K. Winbladh (2013). Analysis of user comments: an approach for software requirements evolution. *Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press.

Ghosh, S., N. Sharma, F. Benevenuto, N. Ganguly and K. Gummadi (2012). Cognos: crowdsourcing search for topic experts in microblogs. *Proceedings of the 35th international ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM.

Group, O. (2006). Business Process Modeling Notation (BPMN) Version 1.0. OMG Final Adopted Specification. *Object Management Group*.

Happel, H.-J. and S. Seedorf (2006). Applications of ontologies in software engineering. *Proc. of Workshop on Sematic Web Enabled Software Engineering* (SWESE) on the ISWC, Citeseer.

Hoda, R., J. Noble and S. Marshall (2011). The impact of inadequate customer collaboration on self-organizing Agile teams. *Information and Software Technology* **53**(5): 521-534.

Holsti, O. R. (1969). Content analysis for the social sciences and humanities.

Horridge, M., H. Knublauch, A. Rector, R. Stevens and C. Wroe (2004). A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0." University of Manchester.

Hosseini, M., J. Moore, M. Almaliki, A. Shahri, K. Phalp and R. Ali (2015). Wisdom of the crowd within enterprises: Practices and challenges. *Computer Networks* **90**: 121-132.

Hosseini, M., K. Phalp, J. Taylor and R. Ali (2014). The four pillars of crowdsourcing: A reference model. *Eighth International Conference on Research Challenges in Information Science,* IEEE.

Hosseini, M., K. Phalp, J. Taylor and R. Ali (2014). Towards crowdsourcing for requirements engineering.

Hosseini, M., A. Shahri, K. Phalp, J. Taylor, R. Ali and F. Dalpiaz (2015). Configuring crowdsourcing for requirements elicitation. *9th International Conference on Research Challenges in Information Science (RCIS)*, IEEE.

Hu, M. and B. Liu (2004). Mining opinion features in customer reviews. *Proceedings of the 19th National Conference on Artifical Intelligence*. San Jose, California, AAAI Press**:** 755-760.

Huh, J., L. Jones, T. Erickson, W. A. Kellogg, R. K. Bellamy and J. C. Thomas (2007). BlogCentral: the role of internal blogs at work. *CHI'07 extended abstracts on Human Factors in Computing Systems*, ACM.

International, QSR. (Copyright © 1999-2014) . NVivo10-Getting-Started-Guide. Retrieved 20 March 2015, from http://download.qsrinternational.com/Document/NVivo10/NVivo10-Getting-Started-Guide.pdf.

Jarke, M., P. Loucopoulos, K. Lyytinen, J. Mylopoulos and W. Robinson (2011). The brave new world of design requirements. *Information Systems* **36**(7): 992-1008.

Jiang, J. J., G. Klein, S. P. Wu and T.-P. Liang (2009). The relation of requirements uncertainty and stakeholder perception gaps to project management performance. *Journal of Systems and Software* **82**(5): 801-808.

Jick, T. D. (1979). Mixing qualitative and quantitative methods: Triangulation in action. *Administrative Science Quarterly* **24**(4): 602-611.

Joffe, H. and L. Yardley (2004). Content and thematic analysis. *Research Methods for Clinical and Health Psychology* **56**: 68.

Johnson, M. and S. Hyysalo (2012). Lessons for participatory designers of social media: long-term user involvement strategies in industry. *Proceedings of the 12th Participatory Design Conference*: Research Papers-Volume 1, ACM.

Kajko-Mattsson, M. (2005). A survey of documentation practice within corrective maintenance. *Empirical Software Engineering* **10**(1): 31-55.

Kang, K. C., S. Kim, J. Lee, K. Kim, E. Shin and M. Huh (1998). FORM: A feature- oriented reuse method with domain- specific reference architectures. *Annals of Software Engineering* **5**(1): 143-168.

Kanstrup, A. M. (2012). A small matter of design: an analysis of end users as designers. *Proceedings of the 12th Participatory Design Conference*: Research Papers-Volume 1, ACM.

Karel Vredenburg, Ji-Ye Mao, Paul W. Smith and T. Carey (2002). A survey of user-centered design practice. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Minneapolis, Minnesota, USA, ACM**:** 471-478.

Kehagias, D. D., I. Papadimitriou, J. Hois, D. Tzovaras and J. Bateman (2008). A methodological approach for ontology evaluation and refinement. In *ASK-IT Final Conference*.

Kensing, F. and J. Blomberg (1998). Participatory design: Issues and concerns. *Computer Supported Cooperative Work* (CSCW) **7**(3-4): 167-185.

Kitchenham, B. A., S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam and J. Rosenberg (2002). Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering* **28**(8): 721-734.

Kittur, A., E. H. Chi and B. Suh (2008). Crowdsourcing user studies with Mechanical Turk. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM.

Kittur, A. and R. E. Kraut (2008). Harnessing the wisdom of crowds in wikipedia: quality through coordination. *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work*, ACM.

Knauss, A. (2012). On the usage of context for requirements elicitation: End-user involvement in IT ecosystems. 20th International *Requirements Engineering Conference (RE)*, IEEE.

Komarov, S., K. Reinecke and K. Z. Gajos (2013). Crowdsourcing performance evaluations of user interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Paris, France, ACM**:** 207-216.

Kontio, J., L. Lehtola and J. Bragge (2004). Using the focus group method in software engineering: obtaining practitioner and user experiences. I*n Proceedings of the International Symposium on Empirical Software Engineering (ISESE '04)*. IEEE. , Redondo Beach, CA, USA.

Kotonya, G. and I. Sommerville (1998). Requirements engineering: processes and techniques, *Wiley Publishing*.

Krogstie, J., K. Lyytinen, A. L. Opdahl, B. Pernici, K. Siau and K. Smolander (2004). Research areas and challenges for mobile information systems. *International Journal of Mobile Communications*, *2*(3), pp.220-234.

Kuhn, T. (2014). A survey and classification of controlled natural languages. *Computational Linguistics* **40**(1): 121-170.

Law, E. L.-C. and P. v. Schaik (2010). Editorial: Modelling user experience - An agenda for research and practice. *Interacting with Computers*, *22*(5), pp.313-322.

Lazar, J., J. H. Feng and H. Hochheiser (2010). Research Methods in Human-Computer Interaction, *John Wiley & Sons*.

Leotta, M., F. Ricca, G. Antoniol, V. Garousi, J. Zhi and G. Ruhe (2013). A pilot experiment to quantify the effect of documentation accuracy on maintenance tasks. *29th International Conference on Software Maintenance* (ICSM), IEEE.

Lientz, B. P. and E. B. Swanson (1981). Problems in application software maintenance. *Communications of the ACM* **24**(11): 763-769.

Lientz, B. P., E. B. Swanson and G. E. Tompkins (1978). Characteristics of application software maintenance. *Communications of the ACM* **21**(6): 466-471.

Liu, B., M. Hu and J. Cheng (2005). Opinion observer: analyzing and comparing opinions on the Web. *Proceedings of the 14th international conference on World Wide Web*. Chiba, Japan, ACM**:** 342-351.

Liu, D., R. G. Bias, M. Lease and R. Kuipers (2012). Crowdsourcing for usability testing. *Proceedings of the American Society for Information Science and Technology* **49**(1): 1-10.

Lu, C.-J. and S. W. Shulman (2008). Rigor and flexibility in computer-based qualitative research: Introducing the Coding Analysis Toolkit. *International Journal of Multiple Research Approaches* **2**(1): 105-117.

Maalej, W., H.-J. Happel and A. Rashid (2009). When users become collaborators: towards continuous and context-aware user input. *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, ACM.

Maalej, W. and D. Pagano (2011). On the socialness of software. *Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC)*, IEEE.

Marra, R. M., J. L. Moore and A. K. Klimczak (2004). Content analysis of online discussion forums: A comparative analysis of protocols. *Educational Technology Research and Development* **52**(2): 23-40.

Miles, M. B. and A. M. Huberman (1994). Qualitative data analysis: An Expanded Sourcebook, Sage.

Mistrik, I., N. Ali, R. Kazman, J. Grundy and B. Schmerl (2016). Managing Trade-offs in Adaptable Software Architectures, Morgan Kaufmann.

Moodle. (2016). Adding a new course. Retrieved 14 November, 2016, from https://docs.moodle.org/31/en/Adding_a_new_course#Using_an_existing_course_as_a_template.

Moodle. (2016). Category Enrolments. Retrieved 18 November, 2016, from https://docs.moodle.org/31/en/Category_enrolments.

Moodle. (2016). Course Settings. Retrieved 14 November, 2016, from https://docs.moodle.org/31/en/Course_settings.

Moodle. (2016). Moodle Courses. Retrieved 8 November, 2016, from https://docs.moodle.org/31/en/Courses.

Moodle. (2016). Upload Courses. Retrieved 13 November, 2016, from https://docs.moodle.org/31/en/Upload_courses.

Noy, N. F. and D. L. McGuinness (2001). Ontology development 101: A guide to creating your first ontology, *Stanford Knowledge Systems Laboratory Technical Report* KSL-01-05 and *Stanford Medical Informatics Technical Report* SMI-2001-0880, Stanford, CA.

Onwuegbuzie, A. J. (2003). Effect sizes in qualitative research: A prolegomenon. *Quality and Quantity* **37**(4): 393-409.

Orb, A., L. Eisenhauer and D. Wynaden (2001). Ethics in qualitative research. *Journal of Nursing Scholarship* **33**(1): 93-96.

Pagano, D. (2011). Towards systematic analysis of continuous user input. *Proceedings of the 4th international workshop on Social software engineering.* Szeged, Hungary, ACM**:** 6-10.

Pagano, D. and B. Brügge (2013). User involvement in software evolution practice: a case study. *Proceedings of the 2013 International Conference on Software Engineering*. San Francisco, CA, USA, IEEE Press**:** 953-962.

Pagano, D. and W. Maalej (2013). User feedback in the appstore: An empirical study. *21st International Requirements Engineering Conference* (RE), IEEE.

Phalp, K., A. Adlem, S. Jeary, J. Vincent and J. Kanyaru (2011). The role of comprehension in requirements and implications for use case descriptions. *Software Quality Journal* **19**(2): 461-486.

Pourshahid, A., D. Amyot, L. Peyton, S. Ghanavati, P. Chen, M. Weiss and A. J. Forster (2009). Business process management with the user requirements notation. *Electronic Commerce Research* **9**(4): 269-316.

Press, W. (2015). Roles and Capabilities. Retrieved 5 May, 2015, from https://codex.wordpress.org/Roles_and_Capabilities.

Rajlich, V. (2014). Software evolution and maintenance. *Proceedings of the on Future of Software Engineering*, ACM.

Robbins, J. E. (2004). Feature Specifications. Software Tools and Methods Course Retrieved 15 December 2016, from http://www.jrobbins.org/ics121s04/lesson-feature-specs.html.

Salehie, M. and L. Tahvildari (2009). Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and adaptive Systems* (TAAS) **4**(2): 14.

Schneider, K. (2011). Focusing spontaneous feedback to support system evolution. *19th IEEE International Requirements Engineering Conference (RE)*, Trento, Italy.

Schöndienst, V., H. Krasnova, O. Günther and D. Riehle (2011). Micro-Blogging Adoption in the Enterprise: An Empirical Analysis: 931-940.

Seyff, N., F. Graf and N. Maiden (2010). Using Mobile RE Tools to Give End-Users Their Own Voice. *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*. Sydney, Australia, IEEE Computer Society**:** 37-46.

Shahri, A., M. Hosseini, K. Phalp, J. Taylor, and R. Ali (2014). Towards a Code of Ethics for Gamification at Enterprise. *In IFIP Working Conference on The Practice of Enterprise Modeling* (pp.235-245). Springer International Publishing.

Shahri, A., M. Hosseini, K. Phalp, J. Taylor, and R. Ali (2016). Exploring and conceptualising software-based motivation within enterprise. *In IFIP Working Conference on The Practice of Enterprise Modeling* (pp. 241-256). Springer International Publishing

Shearer, R., B. Motik and I. Horrocks (2008). HermiT: A Highly-Efficient OWL Reasoner. OWLED.

Sherief, N., W. Abdelmoez, K. Phalp and R. Ali (2015). Modelling users feedback in crowd-based requirements engineering: An empirical study. *IFIP Working Conference on The Practice of Enterprise Modeling*, Springer.

Sherief, N., N. Jiang, M. Hosseini, K. Phalp and R. Ali (2014). Crowdsourcing software evaluation. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. London, England, United Kingdom, ACM**:** 1-4.

Siegemund, K., E. J. Thomas, Y. Zhao, J. Pan and U. Assmann (2011). Towards ontology-driven requirements engineering. Workshop semantic web enabled software engineering at *10th international semantic web conference (ISWC)*, Bonn.

Sillito, J., G. C. Murphy and K. De Volder (2006). Questions programmers ask during software evolution tasks. *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ACM.

Snijders, R., F. Dalpiaz, S. Brinkkemper, M. Hosseini, R. Ali and A. Ozum (2015). REfine: A gamified platform for participatory requirements engineering. *IEEE 1st International Workshop on Crowd-Based Requirements Engineering (CrowdRE)*, Aug 25 (pp. 1-6), IEEE.

Snijders, R., F. Dalpiaz, M. Hosseini, A. Shahri and R. Ali (2014). Crowd-Centric Requirements Engineering. *7th International Conference on Utility and Cloud Computing (UCC),* IEEE.

Sommerville, I. (2006). Software Engineering: (Update) (8th Edition) (International Computer Science), Addison-Wesley Longman Publishing Co., Inc.

Souza, V. E. S., A. Lapouchnian, K. Angelopoulos and J. Mylopoulos (2013). Requirements-driven software evolution. *Computer Science-Research and Development* **28**(4): 311-329.

Spinuzzi, C. (2005). The methodology of participatory design. *Technical Communication* **52**(2): 163-174.

Stemler, S. (2001). An overview of content analysis. Practical assessment, research & evaluation **7**(17): 137-146.

Stolee, K. T. and S. Elbaum (2010). Exploring the use of crowdsourcing to support empirical studies in software engineering. P*roceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. Bolzano-Bozen, Italy, ACM**:** 1-4.

Surowiecki, J. (2005). The wisdom of crowds, Anchor.

Svee, E.-O., C. Giannoulis and J. Zdravkovic (2011). Modeling business strategy: A consumer value perspective. *The Practice of Enterprise Modeling*, Springer**:** 67-81.

Turner III, D. W. (2010). Qualitative interview design: A practical guide for novice investigators. *The Qualitative Report* **15**(3): 754.

University, Bournemouth. (2017). BU Research Blog | Research Ethics | Bournemouth University. Retrieved 9 January, 2017, from http://blogs.bournemouth.ac.uk/research/researcher-toolbox/research-ethics/.

Vaismoradi, M., H. Turunen and T. Bondas (2013). Content analysis and thematic analysis: Implications for conducting a qualitative descriptive study. *Nursing & Health Sciences* **15**(3): 398-405.

Wang, H., Y. Wang and J. Wang (2014). A participant recruitment framework for crowdsourcing based software requirement acquisition. *9th International Conference on Global Software Engineering (ICGSE)*, IEEE.

Würsch, M., G. Ghezzi, G. Reif and H. C. Gall (2010). Supporting developers with natural language queries. *32nd International Conference on Software Engineering*, 2010, IEEE.

Yeoh, W. and A. Koronios (2010). Critical success factors for business intelligence systems. *Journal of Computer Information Systems* **50**(3): 23-32.

Yu, E. S. (2009). Social Modeling and i*. Conceptual Modeling: Foundations and Applications. T. B. Alexander, K. C. Vinay, G. Paolo and S. Y. Eric, Springer-Verlag**:** 99-121.

Zimmermann, T., A. Zeller, P. Weissgerber and S. Diehl (2005). Mining version histories to guide software changes. *IEEE Transactions on Software Engineering* **31**(6): 429-445.

Zin, A. M. and N. C. Pa (2009). Measuring communication gap in software requirements elicitation process. *Proceedings of the 8th WSEAS International Conference on Software engineering, parallel and distributed systems*. Cambridge, UK, World Scientific and Engineering Academy and Society (WSEAS)**:** 66-71.

10. Appendices

10.1 Appendix 1: Sample Ethics Documents

10.1.1 Participant Information Sheet for PD Study

The title of the research project: Designing a Collaborative Framework for Feedback Acquisition and Communication to Support Software Maintenance

Invitation

You are being invited to take part in this research project conducted by Nada Hany Sherief, a postgraduate researcher, in the Department of Computing and Informatics, Faculty of Science & Technology, Bournemouth University, UK. Before you decide, it is important for you to understand why the research is being done and what it will involve. Please take time to read the following information carefully and discuss it with others if you wish. Ask us if there is anything that is not clear or if you would like more information. Take time to decide whether or not you wish to take part.

What is the purpose of the project?

Our aim is to develop a collaborative feedback framework to systematically support software engineers at runtime (i.e. while the software is already in use). This includes employing previously established feedback structures and devising mechanisms that makes it easy for engineer to interpret the users' feedback and benefit from it in the software change management tasks during the maintenance phase. Additionally, this engineering framework aims at benefiting from the collective structured feedback and incorporating potential requirements models to link the space of the business to the space of users and their understanding of the system to keep the requirements information up-to-date, and enables automated reasoning and traceability to derive essential information.

A Participatory Design approach will be used in order to explore the participants' ideas about the facets of the framework. By this, the end product is developed hand in hand with the actual intended audience and thus leads to better results and experiences, as the method can give clear insight into their vocabulary, priorities, and the things they value.

Why have I been chosen?

This is an open call that aims to reach those who feel they can contribute to the research by sharing their experience with technology, utilizing our mechanisms and giving us feedback on them.

Do I have to take part?

It is up to you to decide whether or not to take part. If you do decide to take part, you will be given this information sheet to keep (and be asked to sign a participant agreement form). You can withdraw at any time, up to the point where the data are processed and become anonymous, so your identity cannot be determined, without it affecting any benefits that you are entitled to in any way. You do not have to give a reason. Deciding to take part or not will not adversely affect you.

What would taking part involve?

As a participant in this project, there will be some activities to undertake. First, you will be asked to attend an introductory session to familiarize you with the project, its aims and the needed background information that will be used during the study, and it includes research results reached so far that will be utilized in the study.

Furthermore, during the design session the participants will be provided with aiding tools such as mock-ups for the acquisition tool components to test how they would utilize the components to provide meaningful feedback without hindering their experience. Also, they will be provided with fictional scenarios about software problems to help them immerse in the task. Finally, they will be asked to give insights by answering sets of questions, to develop this engineering framework in a manner that makes it useful and informative for engineers during the maintenance phase.

What are the advantages and possible disadvantages or risks of taking part?

Whilst there are no immediate benefits for those people participating in the project, our goal is designing a framework that aligns with the software change process tasks and at the same time gives customers (i.e. end-users) a voice in the continuous runtime evaluation of software. That is, customers' evaluation feedback would mainly communicate their opinion on the role of the system in meeting their requirements leading to better acceptance of the software. Also, it will provide valuable inputs for engineers that will inform their tasks during the maintenance phase, which can save much time and effort for them.

Will my taking part in this study be kept confidential?

All the information that we collect about you during the course of the research will be kept strictly confidential. You will not be able to be identified in any reports or publications. All data relating to this study will be kept for 5 years on a BU password protected secure network.

What type of information will be sought from me and why is the collection of this information relevant for achieving the research project's objectives?

The data collected are not of a sensitive nature, it covers insights and opinions on the facets that will be explored, that express their vocabulary, priorities and things they value.

The data that form the basis of the analysis in this study will be:

- Participants designs: photos and recordings of participants' presentations of designs
- Participants design conversations: During the design workshops, a tape recorder was placed at every group and the conversations will be recorded for transcription and analysis.

Will I be recorded, and how will the recorded media be used?

Yes. The recording will help the research team to capture the information that will be sought from you during the session. However, you will be given the right to accept or reject recording the interview. No other use will be made of the recording without your written permission, and no one outside the research team will be allowed access to the original recordings. The audio recordings made during this research will be deleted once transcribed and anonymised. The transcription of the interviews will not include your name or any identifiable information. Instead, each person will be identified by their code (i.e. #usr1, #eng3, etc.).

Contact for further information

If you have any queries about this research, please contact Dr. Raian Ali by email on rali@bournmeouth.ac.uk or by phone on 01202 966682 or by post to:

Dr. Raian Ali
Faculty of Science and Technology
Bournemouth University
BH12 5BB

Complaints

If you have any complaints about this project please contact Professor Christine A. Maggs, Executive Dean of the Faculty of Science and Technology at Bournemouth University at the following address:

Professor Christine A. Maggs
Poole House P308, Bournemouth University, Talbot Campus, Fern Barrow, Poole, BH12 5BB
E-mail: researchgovernance@bournemouth.ac.uk

Thank you for taking the time to read this information sheet, and please do not hesitate to contact me if you have any queries.

10.1.2 Participant Agreement Form for PD Study

Full title of project: Designing a Collaborative Framework for Feedback Acquisition and Communication to Support Software Maintenance

Name, position and contact details of researcher: Nada Hany Sherief, Postgraduate researcher, Department of Computing and Informatics, Faculty of Science & Technology, Bournemouth University. Email: nsherief@bournmeouth.ac.uk

Supervisor: Dr Raian Ali, Department of Computing and Informatics, Faculty of Science & Technology. Email: rali@bournemouth.ac.uk

Please Initial or Tick Here

I have read and understood the participant information sheet for the above research project.	
I confirm that I have had the opportunity to ask questions.	
I understand that my participation is voluntary.	
I understand that I am free to withdraw up to the point where the data are processed and become anonymous, so my identity cannot be determined.	
During the tasks of the study, I am free to withdraw without giving a reason and without there being any negative consequences.	
Should I not wish to answer any particular question(s), complete a test I am free to decline.	
I give permission for members of the research team to have access to my anonymised responses. I understand that my name will not be linked with the research materials, and I will not be identified or identifiable in the outputs that result from the research.	
I understand taking part in the research may include being recorded (audio) but that these recordings will be deleted once transcribed and anonymised.	
I agree to take part in the above research project.	

_____ _____ _____

Name or Initials of the Participant Date Signature

_____ _____ _____

Name or Initials of the Researcher Date Signature

10.2 Appendix 2: Focus Groups Questions

Introduction: This Focus Group is part of a research work we are currently involved in. This research is related to the idea of allowing users to act as collaborators with the software, by giving them a way to describe their feedback about the software's features.

Background: In our work, we are trying to take users' judgment on the software. By judgment we mean users' opinions about the quality of software in reaching their requirements. This paradigm advocates two principles. First, quality has to be evaluated iteratively during the systems operation (i.e. at runtime) so that quality evaluation is kept up-to-date. Second, users are the primary evaluators of quality and their judgment is a primitive driver of configuration. At runtime, users' quality feedback is obtained, so that the software is re-configured to satisfy the users' community.

Questions:

1) Would you please tell us if there is anything you would like us to clarify about the background of our discussion?
2) Would you please introduce yourself to the group?
3) Would you state to us, from your point of view, what are the characteristics of a good feedback? We need you to list (then we'll discuss each point together) the key properties that makes a feedback Meaningful or useful?
4) Suppose there is a tool that enables you to express the structure of your feedback (i.e. the way you would like to give your feedback). As a user, what are your expectations from such tool?
5) What are the key features that could be included to such a tool to encourage your willingness to do such a task/job?
6) Suppose you want to give your feedback about your judgment of a feature you are using in certain software, how would you describe such a feedback? i.e. give an example of the feedback you would like to provide.
7) How would you describe the contextual information in your feedback? By context we mean any spacio-temporal, environment, personal, task, or social contexts, while users were providing their feedback.
8) Suppose you are having troubles with describing your feedback about a feature in the software, would you rely on other users' feedback to suggest/provide feedback that you can use (e.g. from a bank of statements)? How could you benefit from such a recommendation?

10.3 Appendix 3: Forums Analysis Intermediate Results

In this appendix the development of the Intermediate results (i.e. the maps developed starting from the initial template till the final thematic map) of the forums analysis are explained. The initial template is represented in chapter 4, Figure 8, in section 4.4.1, while the final thematic map is represented in Chapter 4, Figure 9, in section 4.4.2 and explained in details.

The first modifications that emerged while capturing actual users' feedback are shown in figure 105, which were:

- Adding under Subject → Methods→ a "Links" code as a new method that helps users communicate in feedback. Users tend to use links specially when providing solutions or suggesting workarounds to other users' problems.
- Adding under Subject → Clarity → Understandable → an "Explanation" code to capture the idea that users tend to provide explanations and definitions of features to increase the understandability of their feedbacks and articulate how they perceive the functionality of a feature.
- Moving the "Feature" code from Structure → Specificity to the Subject→ Specificity to identify the feature to which the feedback topic (Subject) is related.
- Moving the "Feedback Type" code from the Subject →Specificity to the Structure→ Specificity to identify the different feedback types the users are providing in the actual body (structure) of feedback.
- Removing the Structure → "Timing" node→ with both its child nodes "Immediate" and "Delayed", as no phrases where coded there. We have found it not relevant while capturing actual feedbacks from forums. This is because there were never indications of whether the user was giving the feedback at runtime immediately when the problem occurred, or as a delayed feedback. Also, we thought that it should not be considered as a Structure concept, because timing will not affect the structure of a feedback, but rather it may only help the user explaining better if he gave the feedback during the occurrence of the problem.
- Adding under Structure → Measurement → a "Vote" code as we have observed that at times users may reply on others' feedbacks by stating how much they agree or disagree with them.
- Adding more codes into Structure→ Level of Detail → Depth, which are:
    - "Illustrations": captures the users' descriptions in their feedback whether they are textual descriptions of what happened while they were using a certain feature, or they referenced other supporting documentations that provide descriptions of the feature they are giving feedback about.

    - “Try-out”: captures the explanations of the trails that users have undergone in order to solve the problem they are facing but did not result in resolving their issue.
    - “Scenario”: represents how users add steps in their feedback responses that explain a sequence of activities to the users with the problem in order to reach a solution.
- Adding more codes into Structure → Level of Detail → Context, which are:
    - “Personal”: it captures the user mental aspects. It describes information like mood, expertise and stress.
    - “Task”: it captures what the user is doing, such as describing the tasks that the user was involved in when a problem occurred.
- Adding more codes into Structure → Specificity → Feedback Type, which are:
    - “Help Request”: captures the most common feedback type in which users post problems they are having in using a certain feature or resolving a certain problem that occurs while using the software.
    - “Suggestion”: captures the feedback type in which users respond by providing suggestions to the user with the problem that they think might help in resolving their issue.
    - “Solution”: captures the feedback type in which users respond by providing definitive solutions to the user with the problem whether through a scenario they provide or a reference to a link that states these steps.
    - “Enhancement”: captures the feedback type in which users post in their feedback a new feature they would like to have in the software or a change they wish to have in a feature properties that would better fit their needs.
    - “Investigation”: is a response feedback type in which users respond to a feedback by asking further questions to clarify unclear steps or issues with the users who posted the main problem.
    - “Correction”: is a response feedback type where users correct the misunderstanding of feature definitions or feature usages to other users.

Figure 105. Intermediate Thematic Map 1 for Forums Analysis

The second adjustments that emerged after further coding of actual users' feedback are shown in figure 106. At this point these adjustments emerged, because we started noticing a key difference between the template we were working on, which is initially derived from the focus groups and what is happening in the actual feedback mediums. Mainly, in the focus groups users were giving insights on what they wish they could see in the feedback acquisition methods. Many of these insights proved to be important in out forums analysis. However, in order to reach their level of expectations we have to define what actually constitutes a feedback. The adjustments are explained below:

- Removing the "Structure" code from Subject → Clarity. In the focus groups, participants expressed that it would increase the clarity of the feedback if the feedback would have a certain structure. However, this is our main research aim which is ***to study common feedback structures and their pillars***. So using it as a code in the template during our forums analysis was redundant, because there is no defined structure in actual user feedbacks.

- Moving the “Explanation” code from Subject → Clarity → Understandable to Structure → Level of Detail → Depth, and renaming it as “Feature Definitions” to increase the precision of its purpose and narrow down the types of phases coded in it.
- Removing the “Keywords”, “Scaling”, and “Audio” codes from Subject method, because they were never used. Mainly because users use forums to seek help, or requesting new requirement, or asking for changes in existing features, scaling is rarely used. This method contrasts with the main purpose of forums (i.e. users expressing themselves freely with details) as it is shallow and doesn’t give much details. Also, in the feedbacks we have analysed, keywords were never used as a method because there are no standardized keywords for users to use in their feedback on any forum. Moreover, we have never encountered audio feedbacks.
- Removing the “Statistics” and “Experience”, as in the feedbacks we have analysed no numeric statistics on the usefulness of a feedback was used. Also, we have found no measurements of user experience whether it is better or worse relative to changes in requirements. This is because these dimensions should be derived from accumulated users’ feedbacks and should not be left for users’ subjectivity.
- Adding three new codes under Structure→ Specificity→ Relationships, which are:
    - “Agree”: captures the relationship between feedbacks responses where users tend to just agree with what others are saying.
    - “Disagree”: captures the relationship between feedbacks responses where users tend to disagree with what others are saying.
    - “Extend”: capture the relationship between feedback responses where the user agrees/disagrees with others but adds more details in his opinion indicating reasons why he agrees/disagrees, context information, and/or other related problems that occurred to him.

Figure 106. Intermediate Thematic Map 2 for Forums Analysis

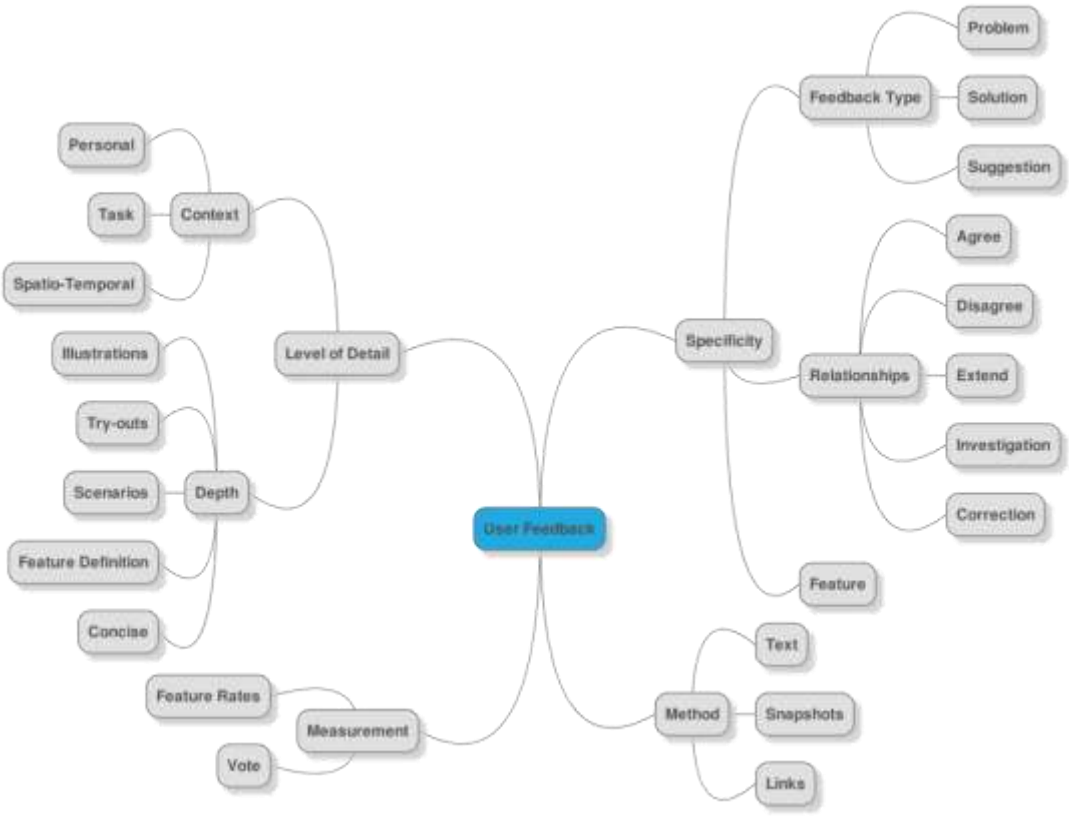

Figure 107. Intermediate Thematic Map 3 for Forums Analysis

Looking at the intermediate template 2 for forums analysis in figure 106, we discussed and came into conclusion that we should remove the level of Structure and Subject thematic areas as shown in Figure 107. These two areas were developed in the final thematic map of the focus groups, where the questions asked were much broader than what we are focusing on in the forums analysis. They were formed to differentiate between the characteristics of a feedback structure (i.e. Structure thematic area), and the context in which users would like to respond to feedback requests (i.e. the Subject thematic area). However, in the forums analysis Subject is irrelevant as there are no feedback requests; instead forums are a community of users seeking help or providing help. Therefore, our main concentration in the forums analysis is to gather the concepts that form an expressive feedback structure, and according to that view we have made the following adjustments:

Removing the “Quality Attribute” from Subject→Specificity. Most users specify clearly the problems in using features, but adding the detail of which aspect of the feature that concerns them was uncommon.

- Removing the “Enhancement” from the Specificity→ Feedback Type. This was done because users rarely provide a clear definition of enhancements they want in the system (i.e. new requirements). Instead they tend to explain the problems they are facing in using the system or performing a certain task.

- Moving the “Measurement”, “Level of Detail”, “Method”, and “Specificity” to connect directly as the main parts of the “User Feedback” node.
- Moving the “Feature” code under User Feedback → Specificity. Therefore, this resulted in a single specificity node that encompasses “feature”, “feedback type”, and “relationships”.
- Removing the “Subject”, and “Structure” thematic areas that are now empty with no codes or categories.
- Moving the “Correction” and “Investigation” codes under “Feedback Types” to “Relationships”. This is because these two codes represent response types to a main feedback post and NOT as an initial feedback type.
- Renaming the “Rates” code under “Measurement” as “Feature Rates”. This is to ensure understandability of the code and that it is applied for features only.
- Renaming the “Help Request” code under “Feedback Type” as “Problem”. This is because it captures the feedbacks that are mainly explaining a problem in detail and asking for help at the end of the post. So we thought it is clearer to name it this way.
- Renaming “Try-outs” code under Level of Detail → Depth as “Trails” for simplicity.
- Adding a “Concise” code under Level of Detail → Depth to capture the short feedback responses that provide no depth and therefore minimum level of detail.
- Adding a “Spatio-Temporal” code as a new context type under Level of Detail→ Context that capture the aspects related to the time and the space. It has attributes like: time, location and place.

Finally, we have reached a more developed thematic map that includes the following changes, and are shown in Figure 108:

- Adding two different codes under Specificity → Feature, which are:
    - “Single”: is a code that captures the feedbacks referring to a single feature of concern.
    - “Group”: is a code that captures the feedbacks referring to a group of related features that work together to perform a certain task.
- Moving the “Relationships” from User Feedback → Specificity to emanate directly from User Feedback. This was done because mainly Relationship types are not specified by the user within his feedback like features and feedback types. Instead, the user can provide feedback responses with several relationships with the initial feedback or other feedbacks along the whole thread.
- Adding several codes to the “Relationships” to capture the different responses and differentiate on how they relate with each other or with the initial feedback, which are:

    - “Mitigation”: this relationship indicates that the user who reported the problem found his own solution, or another user gave him a solution or suggestion that solved his problem.
    - “Justification”: this relationship indicates the reasons why a user offered a certain solution or suggestion in his feedback.
    - “Elaboration”: this relationship is used when a user explains more about a feedback he gave in another post.
    - “Addition”: is used when a user replies on a feedback by confirming or negating the main problem statement in the initial feedback, and adds another separate problem in his feedback.
    - “Verification”: is used when the user who posted the problem gives his opinion on the solutions or suggestions he received.
- Renaming the “Agree” relationship to “Confirmation”, this still indicates when a user only agrees on what others post.
- Renaming the “Disagree” relationship to “Negation”, this still indicates when a user only disagrees on what others post.
- Renaming the “Extend” relationship to “Extension”, and giving a more detailed definition of the types of information it can capture. This relationship is used if the user tried a solution and it solved part of his problem, but this solution led to another dependent problem to emerge.

Figure 108. Intermediate Thematic Map 4 for Forums Analysis

10.4 Appendix 4: Sample PD study Evidence

10.4.1 Toolbox Explanations

In Chapter 6, section 6.2.2.3 a sample toolbox was provided to demonstrate its purpose and design. The tool box in Figure 37 illustrated the Depth types with examples. However, toolboxes for Methods in Figure 109, Context Types in Figure 110, and Attempto Controlled English Rules in Figure 111 were also designed. But to avoid extra elaborations within the text, they are listed in this appendix:

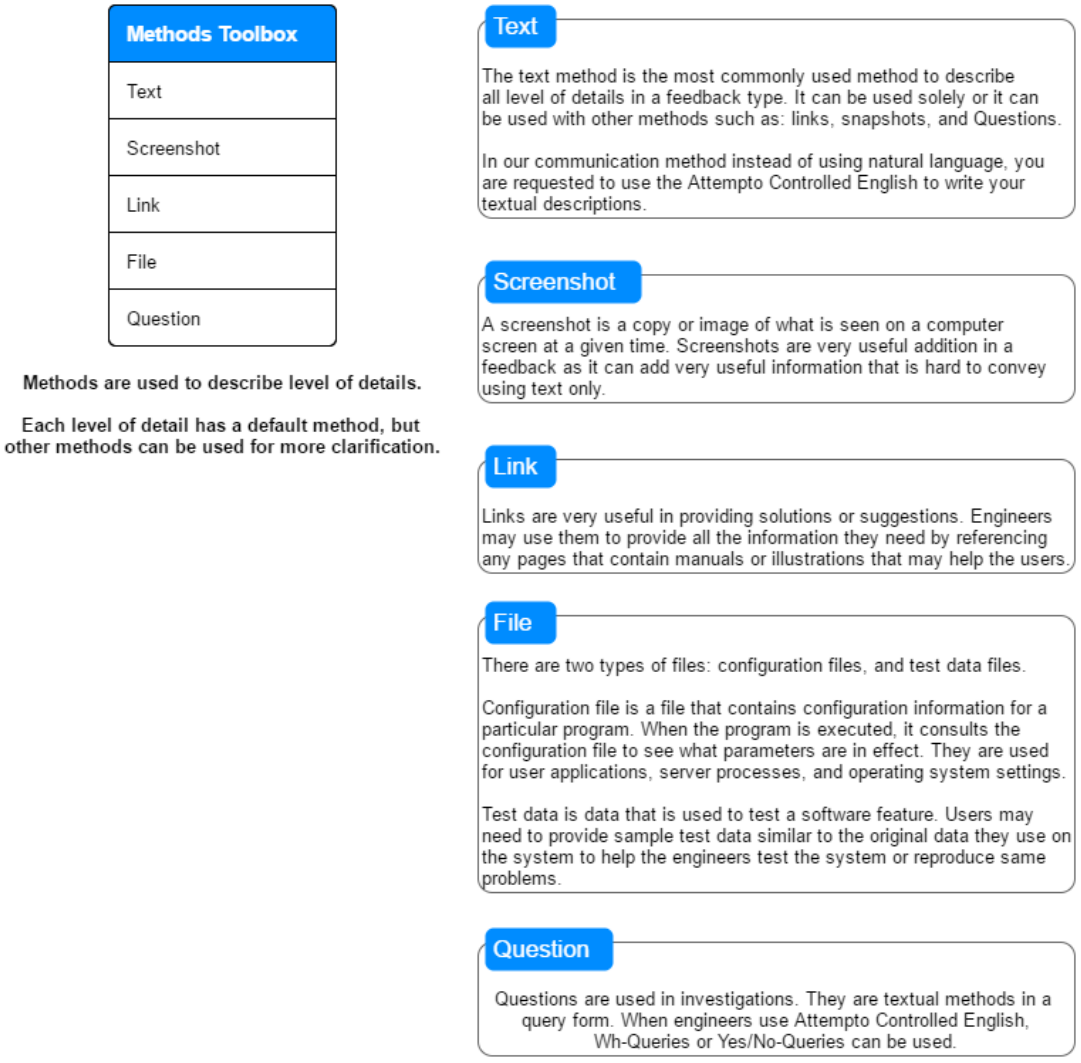

Figure 109. A Toolbox for the Types of Description Methods with Examples.

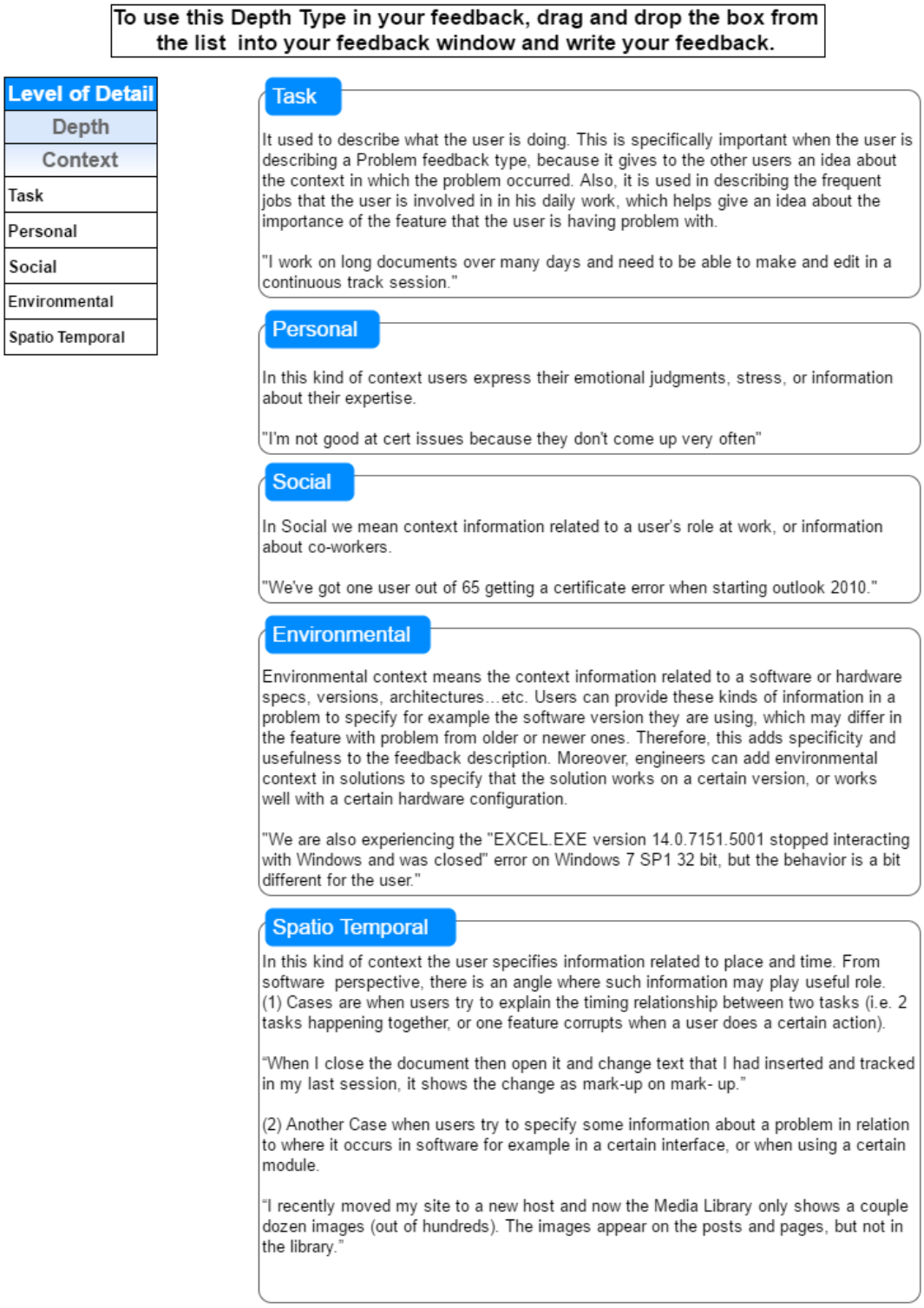

Figure 110. A Toolbox for Context Types with Examples.

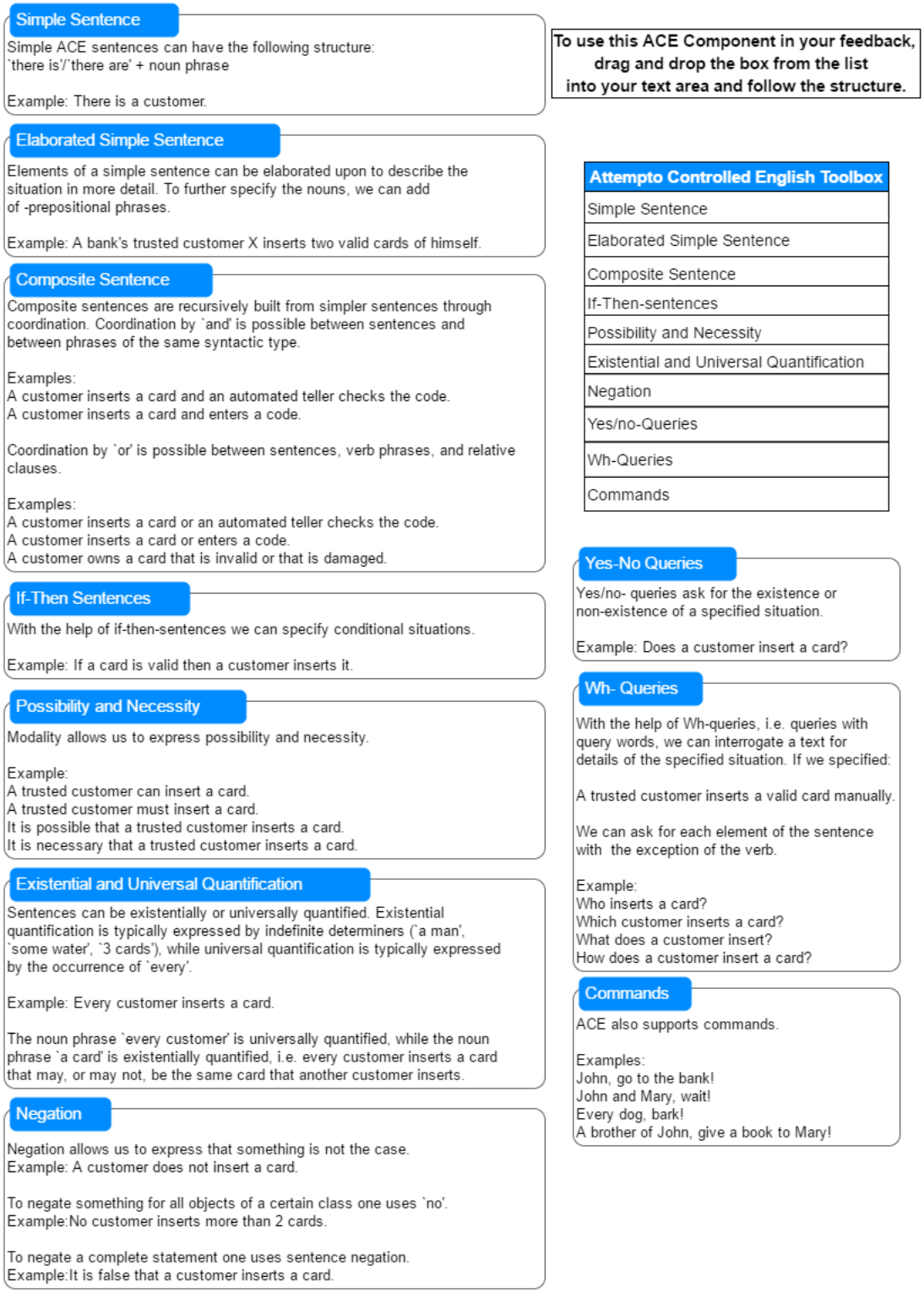

Figure 111. A Toolbox for Attempto Controlled English with Examples.

10.4.2 A Real Sample of Documents for a PD session

In this section a sample communication thread is shown. This thread was explained in detail in chapter 6 section 6.3.1.1.3, where in section 6.3.1.1, selected threads that helped the novel method for feedback acquisition and communication evolve, were explained in detail. In this section one of the used threads with information filled in from participants (both end-user and engineer) is represented. In Figures 112-118 the templates used in the thread are illustrated.

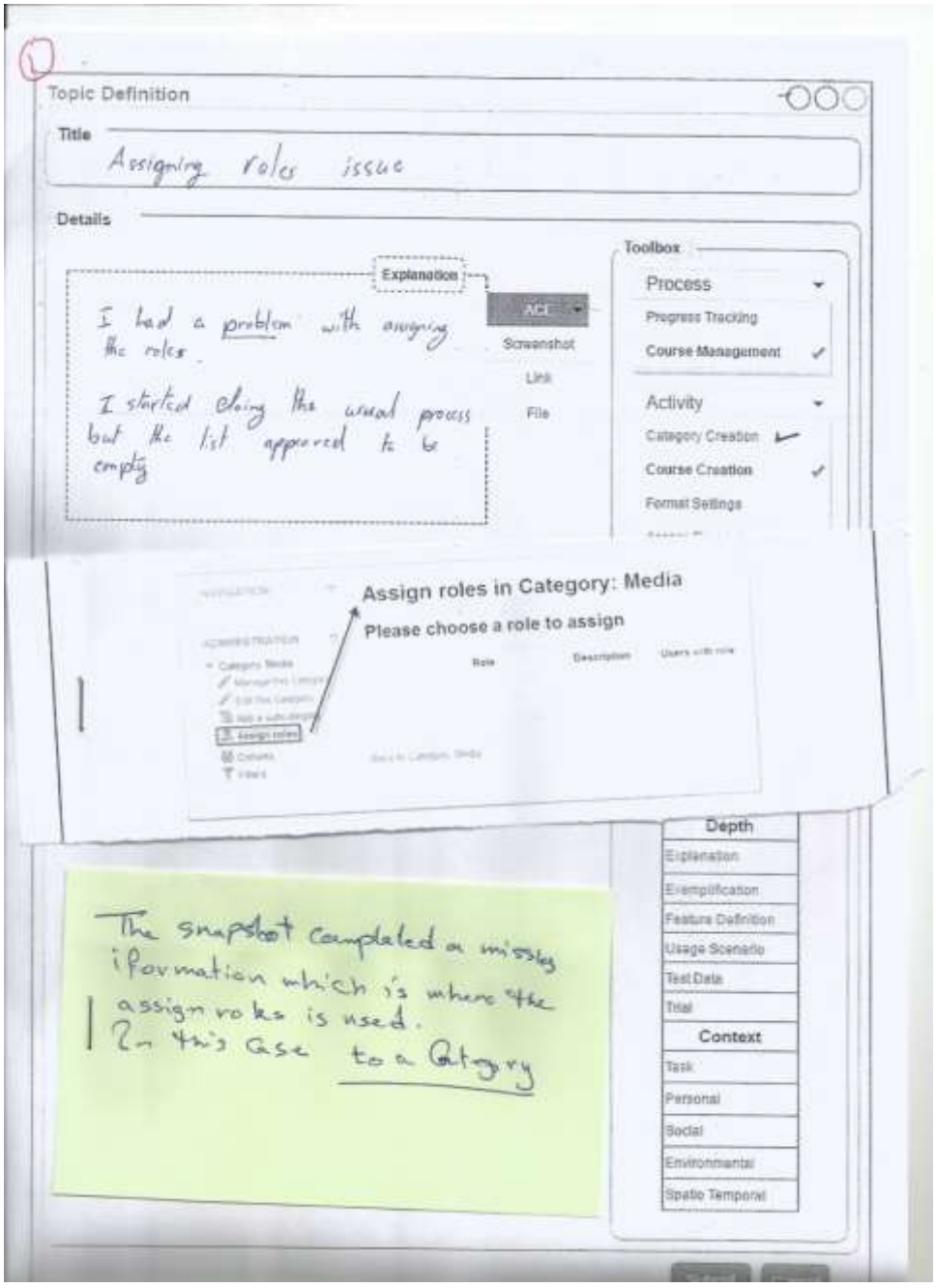

Figure 112. A Real Sample of a Topic Definition Template provided by the End-user Participant

As seen in Figure 112 above that the end-user participant used the screenshot method to provide a Task level of detail in the Topic Definition template. This screenshot was cut from the printed scenario description that was distributed in the beginning of the session. Also sticky notes were used (The pink and green attached notes) to provide extra notes.

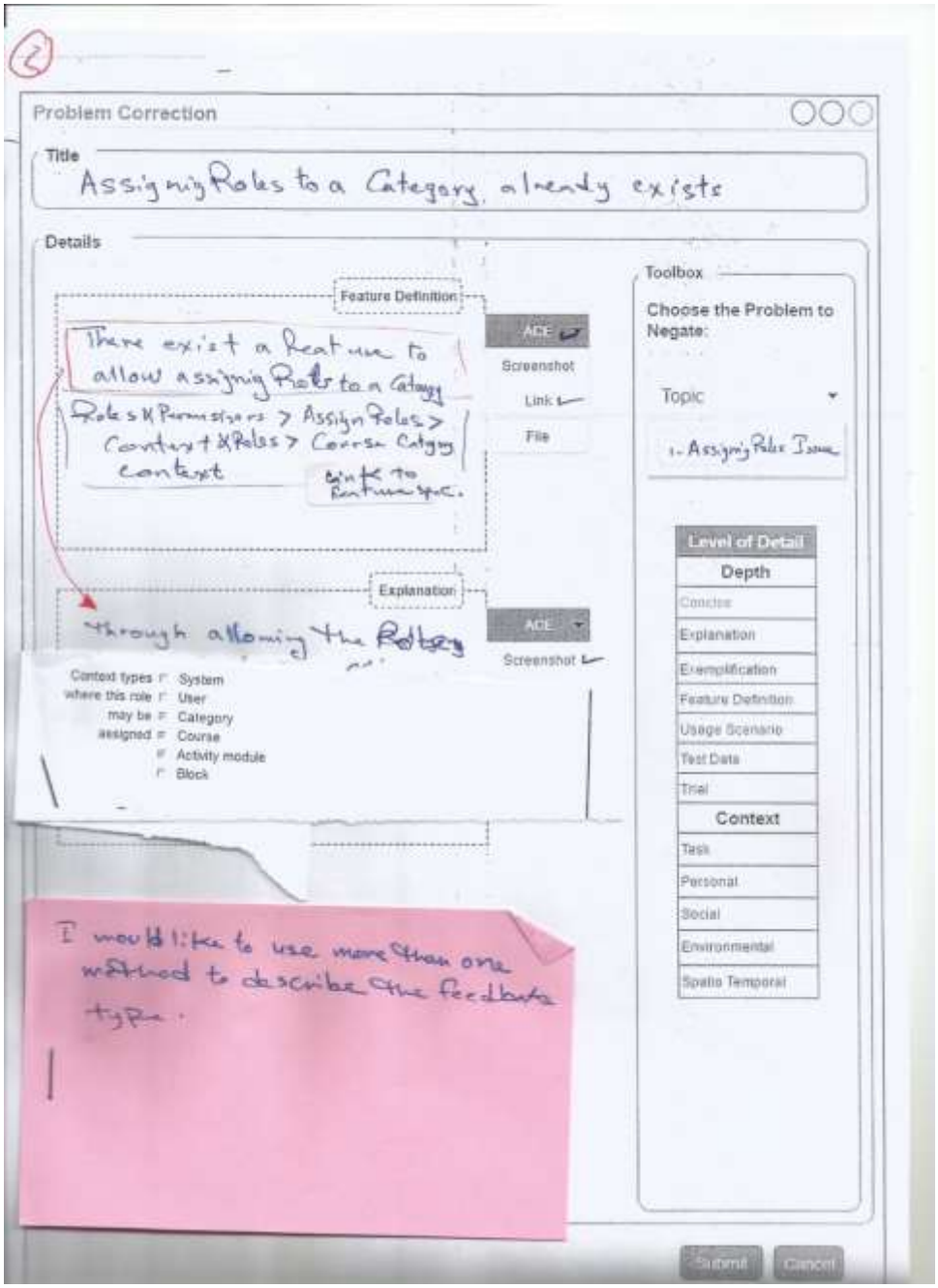

Figure 113. A Real Sample of a Problem Correction Template provided by the Engineer Participant

Also in Figure 113 above that the Engineer participant used both the screenshot and textual methods to provide an Explanation level of detail in the Problem Correction template. This screenshot was cut from the printed scenario description that was distributed in the beginning of the session.

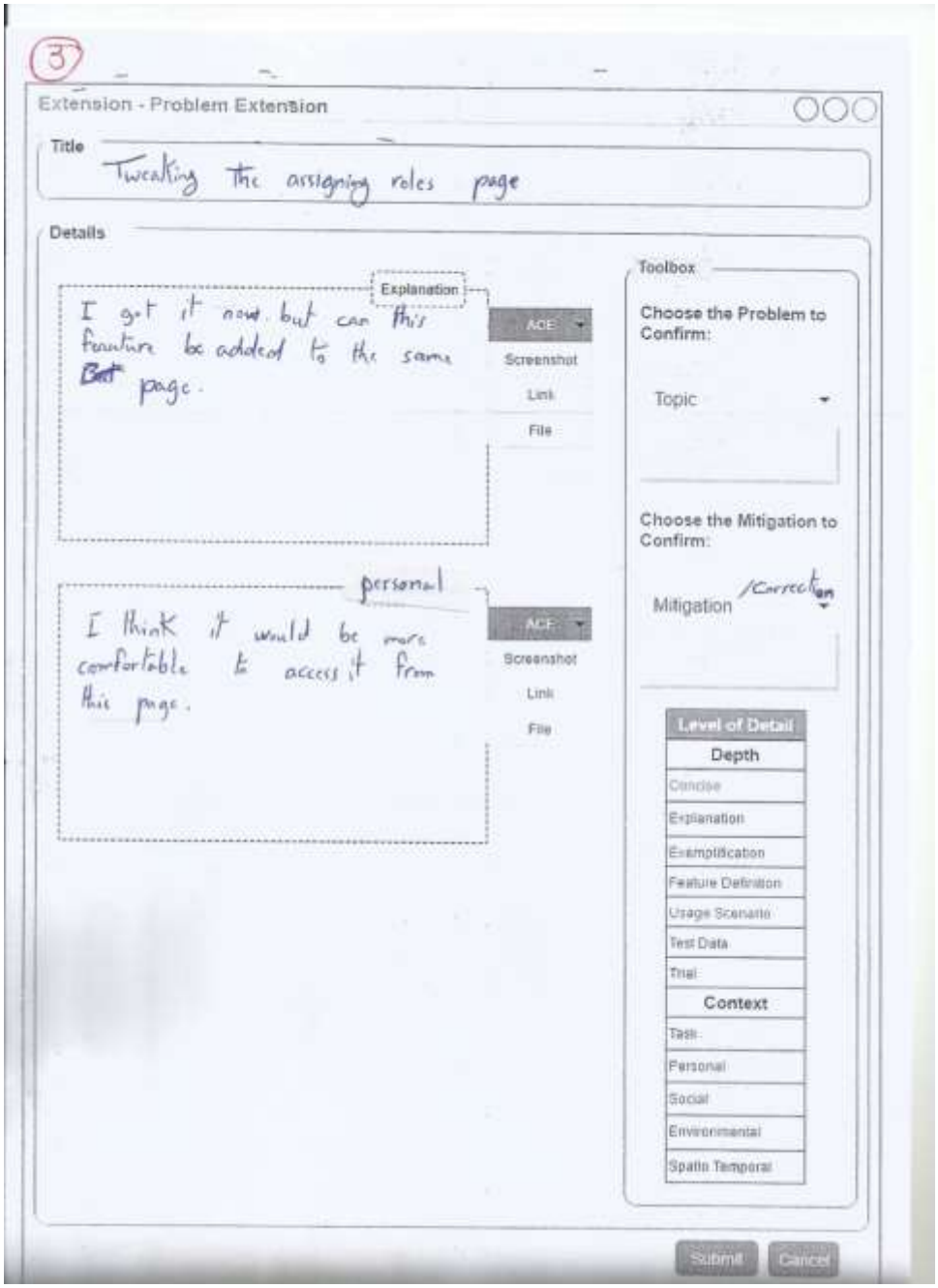

Figure 114. A Real Sample of the New Addition Template provided by the End-user Participant

In figure 114 above, a new feedback type evolved in the session. However, since it is a new feedback type, there was no pre-prepared template for it, so the participant used an existing template (Problem Extension), and he started editing it. For example, he changed one of the default levels of detail and replaced it with a ticket (i.e. tag) to indicate the new type he will be using (i.e. personal context). Also, he added in the right side that the template should be linked to Correction feedback to indicate that it is an enhancement.

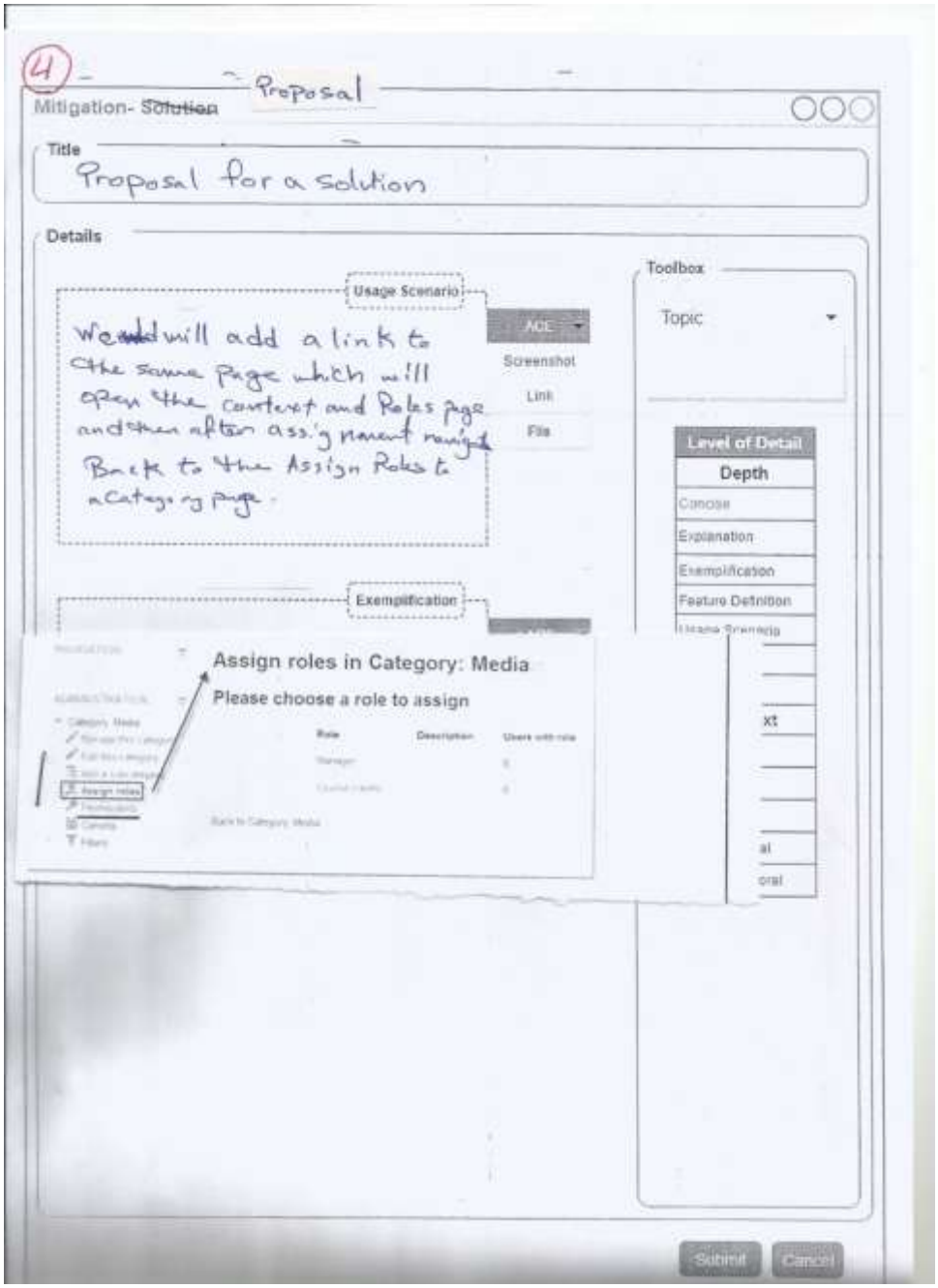

Figure 115. A Real Sample of the New Proposal Template provided by the Engineer Participant

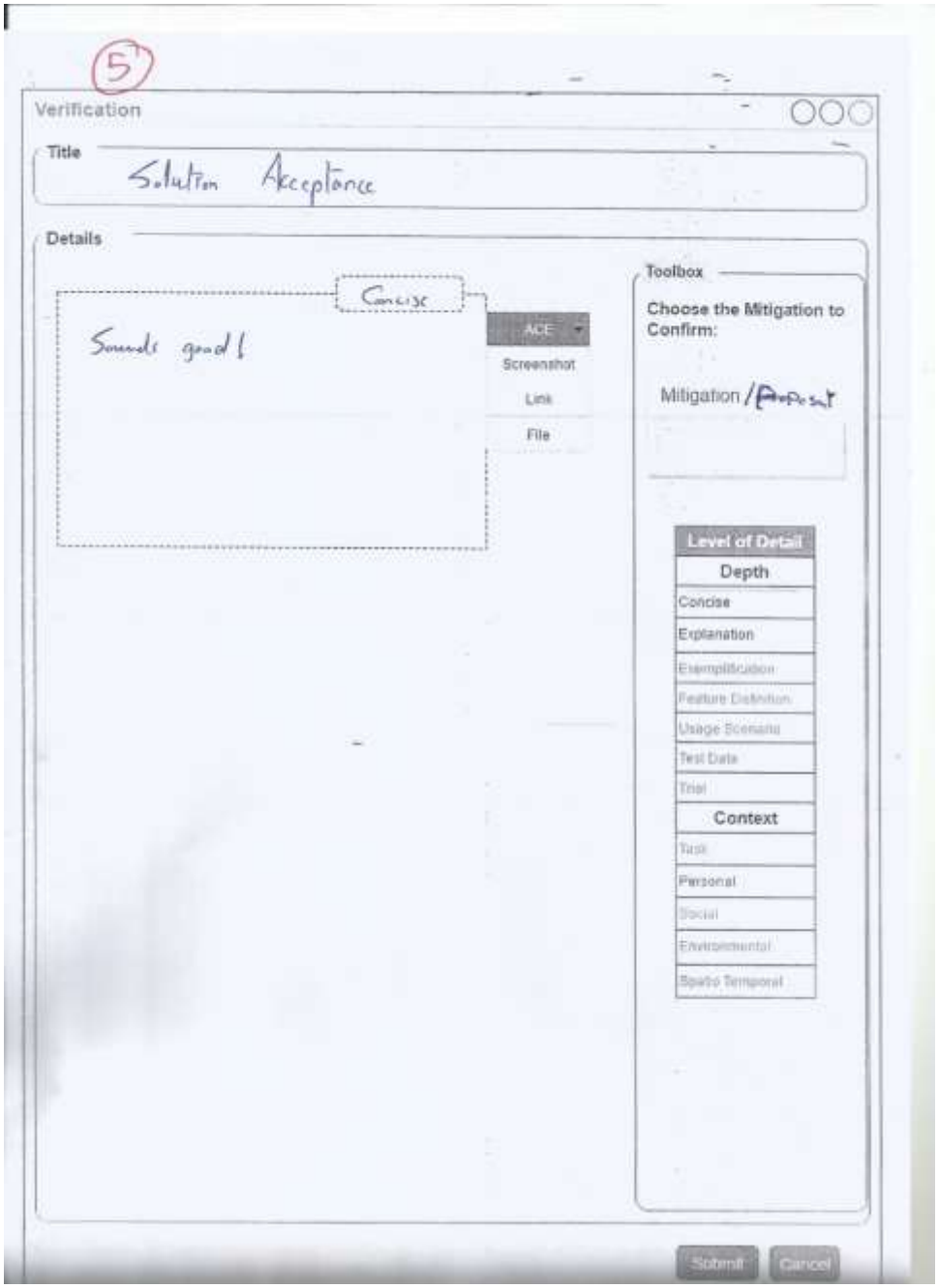

Figure 116. A Real Sample of the Verification Template (Used as Confirmation) provided by the End-user Participant

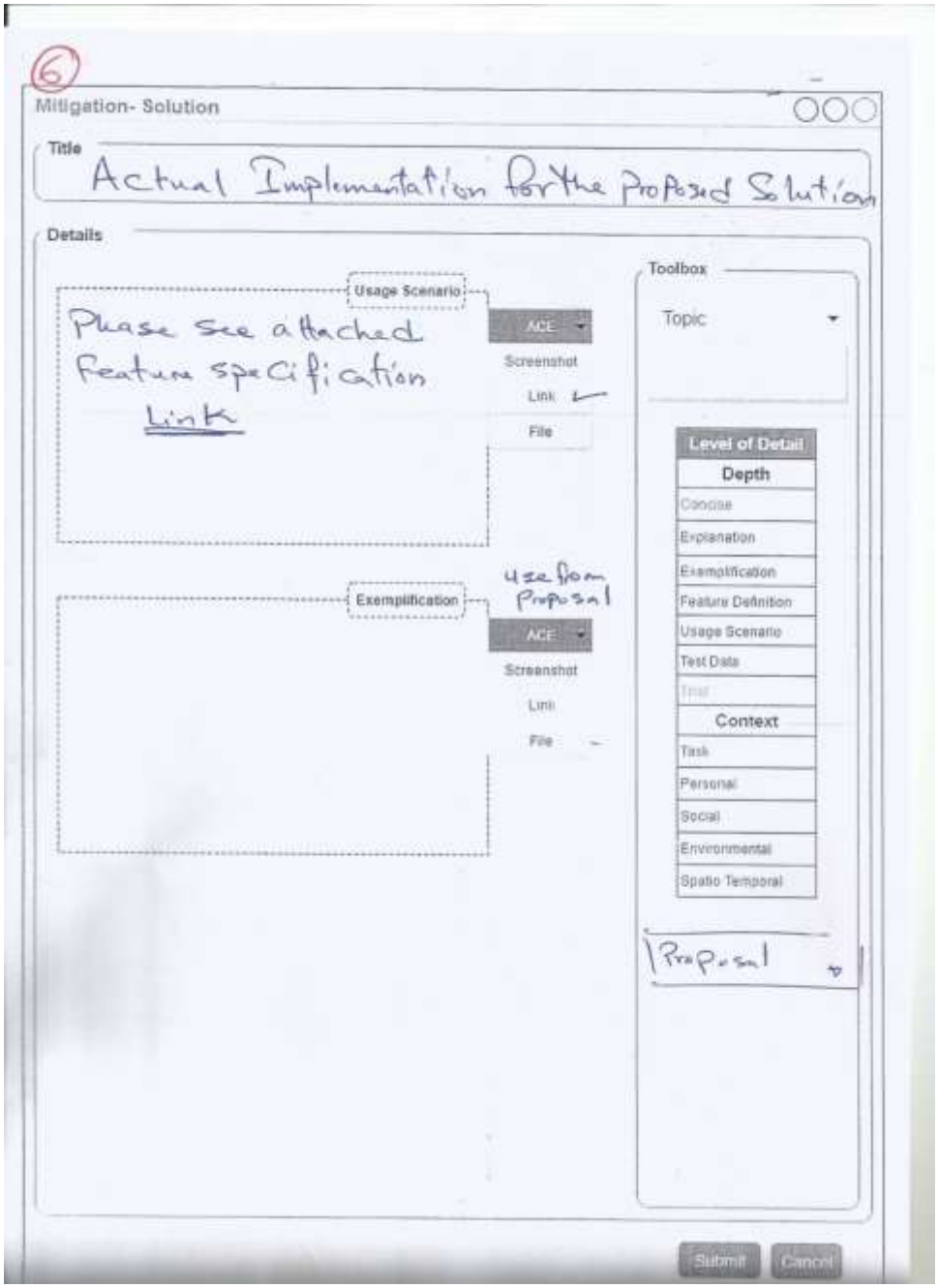

Figure 117. A Real Sample of the Solution Template provided by the Engineer Participant

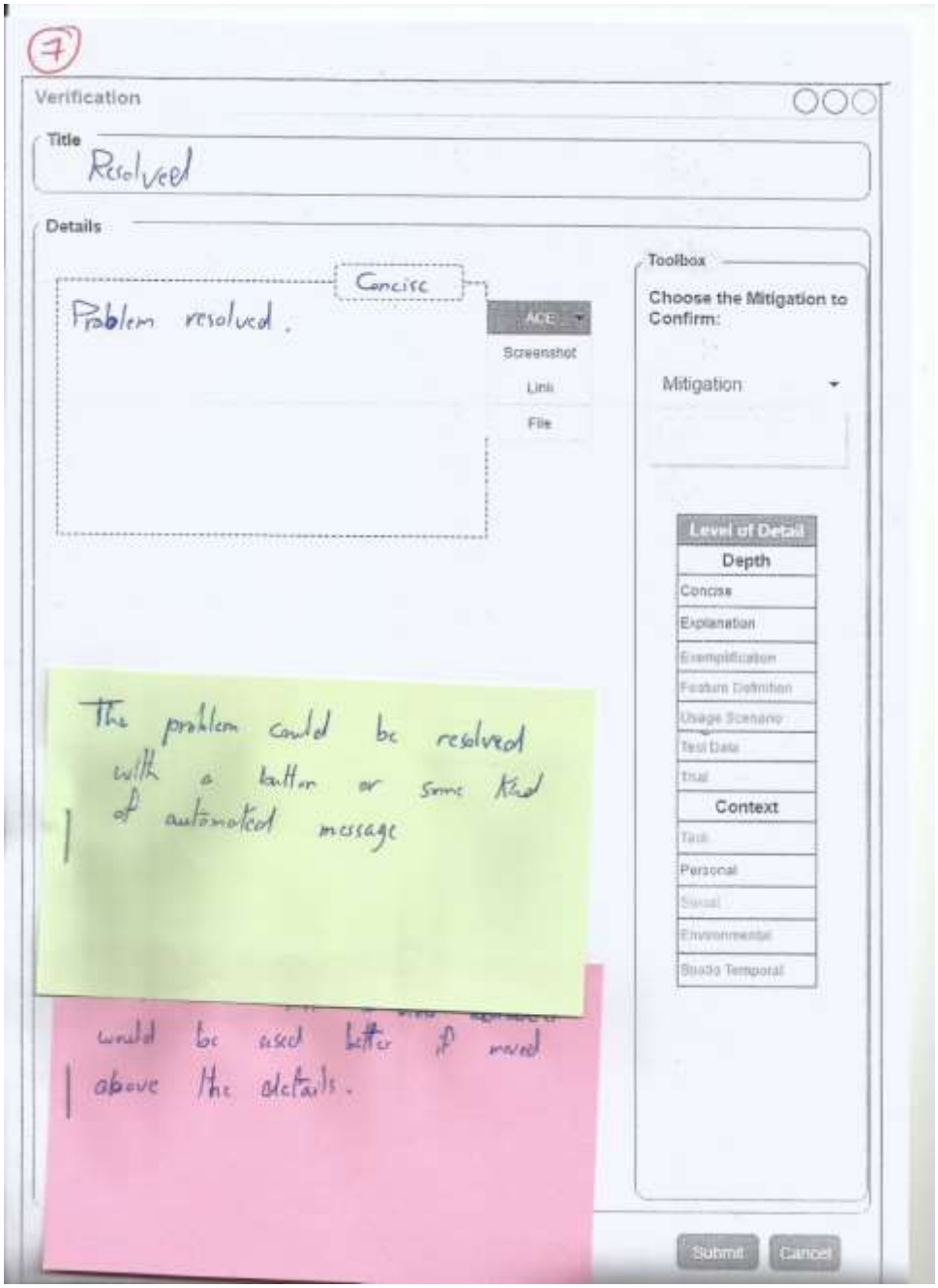

Figure 118. A Real Sample of the Verification Template (Used for final Confirmation) provided by the End-user Participant

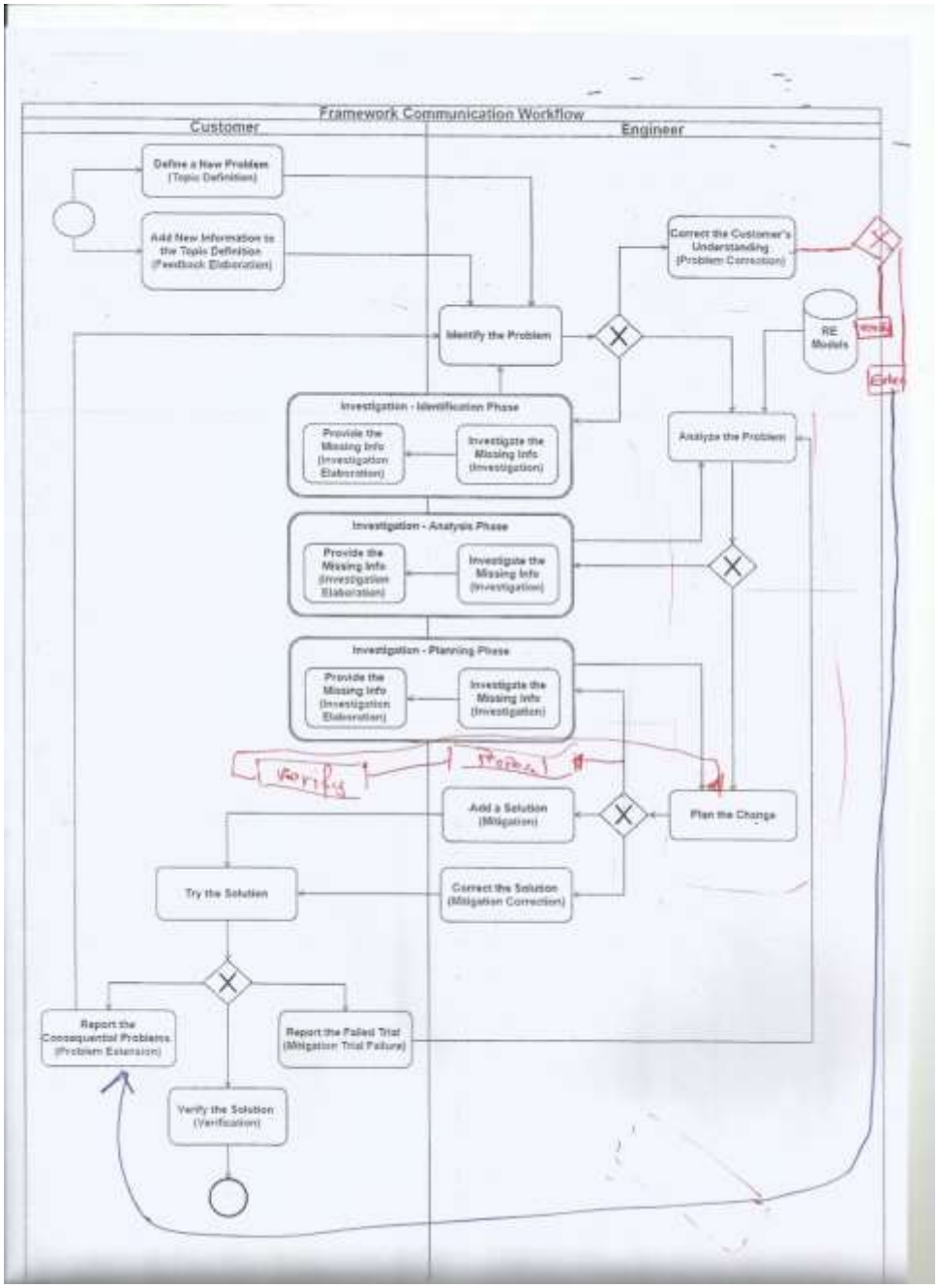

Figure 119. The Updated Feedback Acquisition and Communication Method

Figure 119 above shows the Feedback Acquisition and Communication method design that was distributed on the participants in the session. After the communication took place between them as shown in the above Figures 112-118, they started updating the new method workflow with the feedback types, and new paths that took place.

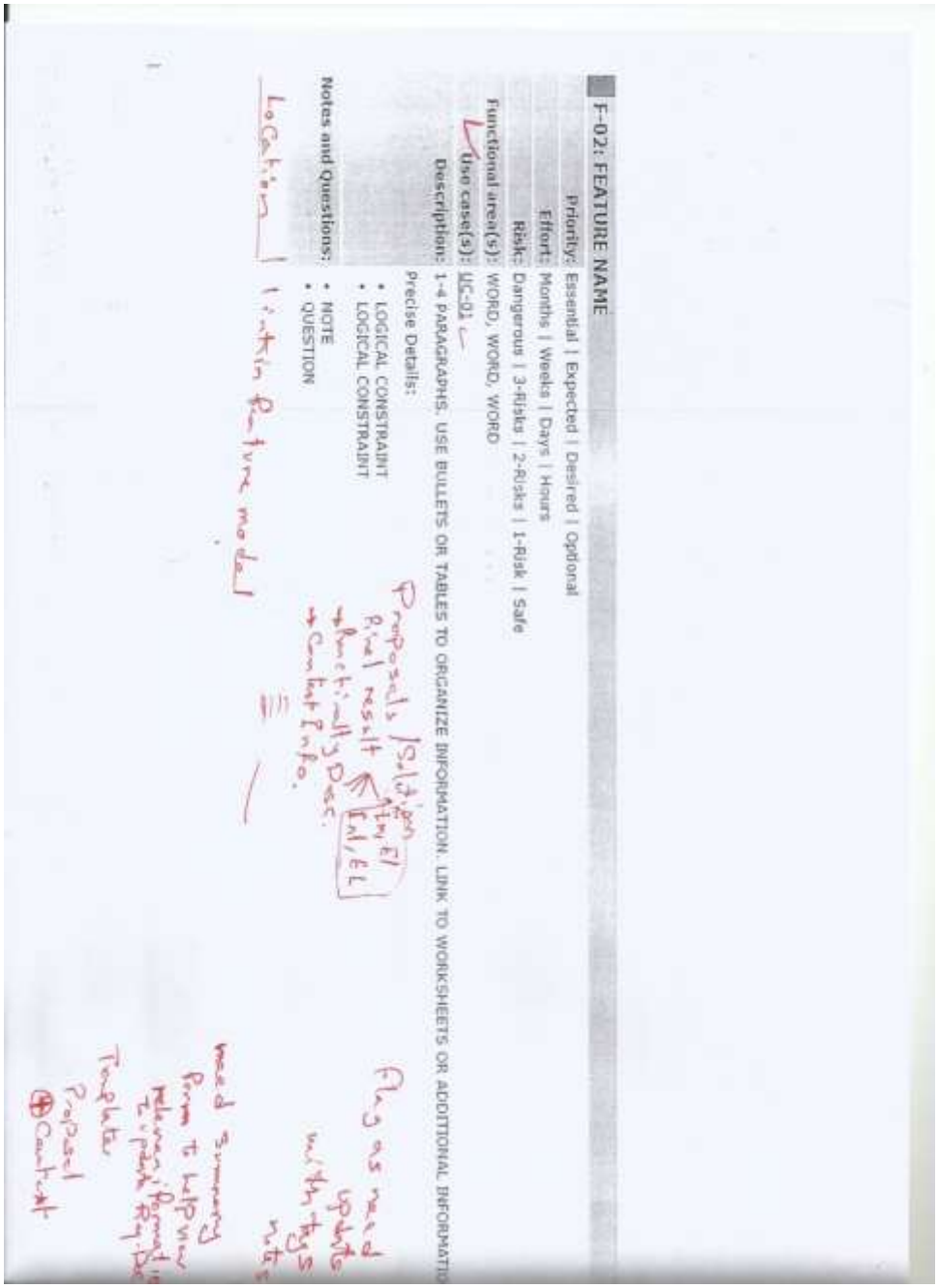

Figure 120. The Feature Specification Template (amended from Robbins 2004) with Participant's updates

Figure 120 above shows the feature specification template that was distributed on the participants with their notes on the sections to add, and how to obtain information to fill these sections.