Graduate School of Natural Science
Master Business Informatics

From user feedback to requirements using chatbots

Master Thesis

Danny Horvath - 6253733

Supervisors:
Dr. Fabiano Dalpiaz
Dr. Sergio Espana Cubillo

1.0 version

Utrecht, November 2019

Abstract

This thesis studies the opportunities of involving users in the requirements engineering process using chatbots. First a semi-structured literature study explores the current field of Crowd Requirements engineering, user feedback and chatbots in general. This literature study leads to a number of initial requirements for using chatbots in the requirements engineering process. Complementary requirements are gathered using a semi-structured interview with the stakeholders at the case study company, leading to an initial requirements list regarding chatbots in the field of requirements engineering. Furthermore, chatbots can help in guiding the users through the bug reporting process by controlling the flow of the conversation and using follow-up questions to further specify information. Therefore, we have designed and evaluated a conversation flow specific for this case. However, we have found that the 5W's used in journalism can be applied to specify the details of a bug report. Finally, we have implemented automatic entity detection in the prototype, this way we were able to further pinpoint the specific bugs and turn them into proto-requirements. After the evaluation the subjects were asked to rate the chatbot using a usability scale, and were interviewed to find out their opinions about using a chatbot for requirements engineering and identifying the strengths and weaknesses of the chatbot. Finally, the resulting bug reports generated by the chatbot were discussed with the developers, they rated the bug reports generated by the chatbot higher or equal to the original bug reports. The most recurring reason for this higher score is the structure that the generated bug reports provide the developers with.

Keywords: Requirements engineering, chatbots, NLP, User feedback

Contents

Contents iii

List of Figures vi

List of Tables vii

1 Introduction 1

2 Case Description 3
2.1 Initial prototype requirements . . . 4

3 Literature Review 6
3.1 Requirements Engineering . . . 6
3.1.1 RE activities . . . 6
3.1.2 Requirements elicitation . . . 7
3.1.3 Requirements analysis . . . 8
3.1.4 Requirements specification . . . 8
3.2 CrowdRE . . . 8
3.2.1 Motivating crowd members . . . 9
3.2.2 Eliciting user feedback . . . 10
3.2.3 Analyzing user feedback . . . 10
3.2.4 Monitoring contextual and usage data . . . 10
3.3 User feedback . . . 11
3.3.1 Feedback classification . . . 12
3.4 Chatbots . . . 13
3.4.1 Interaction model . . . 15
3.4.2 Initiation . . . 16
3.4.3 Intelligence . . . 17
3.4.4 Purpose . . . 17
3.4.5 Environment Dimension . . . 19
3.4.6 Intrinsic Dimension . . . 22
3.4.7 Interaction Dimension . . . 27
3.4.8 Human-Chatbot conversations . . . 31
3.5 Prototype Requirements . . . 32
3.5.1 Environment dimensions . . . 32
3.5.2 Intrinsic dimensions . . . 32
3.5.3 Interaction dimensions . . . 33

4 Research Method 34
4.1 Research questions . . . 34
4.2 Method . . . 35
4.2.1 Problem investigation . . . 36
4.2.2 Treatment design . . . 36

4.2.3 Treatment validation . . . 36
4.2.4 Chatbot purpose . . . 36

5 Conversation flows **39**
5.1 Software Request Structure . . . 39
5.1.1 Problem statement . . . 40
5.1.2 Affected users . . . 40
5.1.3 Bug location . . . 40
5.1.4 Reconstruction scenario . . . 40
5.2 Initial conversation flow . . . 40
5.2.1 Production blocking question . . . 41
5.2.2 Uncovering the problem . . . 41
5.2.3 Question expected end-result . . . 42
5.2.4 Reconstructing the scenario . . . 42
5.2.5 Mapping the urgency . . . 42
5.2.6 Summarize request . . . 42
5.2.7 Rate chatbot . . . 42
5.3 Conversation flow adjustments . . . 43
5.3.1 Question problem adjustments . . . 43

6 Prototype design **44**
6.1 Microsoft LUIS . . . 44
6.1.1 LUIS vs. Rule-based pattern-matching . . . 44
6.1.2 LUIS Development cycles . . . 47
6.2 Microsoft Bot Framework . . . 48
6.2.1 Bot structure . . . 49
6.3 First prototype cycle . . . 50
6.3.1 Bot Design . . . 50
6.3.2 LUIS application design . . . 51
6.3.3 Prototype testing results . . . 52
6.4 Second prototype cycle . . . 53
6.4.1 Bot Design . . . 53
6.4.2 LUIS application design . . . 54
6.4.3 Prototype testing results . . . 55
6.5 Third prototype cycle . . . 55
6.5.1 Bot Design . . . 56
6.5.2 LUIS application design . . . 57
6.5.3 Prototype testing results . . . 58
6.6 Fourth prototype cycle . . . 59
6.6.1 Bot Design . . . 59
6.6.2 LUIS application design . . . 60
6.6.3 Prototype testing results . . . 62
6.6.4 Proto-Requirements . . . 62
6.7 Machine learning analysis . . . 63

7 Prototype evaluation **65**
7.1 Experimental design . . . 65
7.2 Hypotheses . . . 66
7.3 Result analysis . . . 66
7.3.1 SUS score analysis . . . 67
7.3.2 Prototype evaluation . . . 67
7.3.3 Prototype evaluation summary . . . 70

8 Conclusions & Discussion **73**
8.1 Conclusion . . . 73
8.1.1 Research questions . . . 73
8.1.2 Motivating users to provide feedback through a chatbot integrated in a software product (RQ 1) . . . 73
8.1.3 Guiding users in providing informative feedback about a software product using a chatbots (RQ 2) . . . 74
8.1.4 Automatically eliciting proto-requirements from user feedback (RQ 3) . . . 74
8.1.5 Using chatbots integrated in a software application to elicit requirements . . . 74
8.2 Lessons learned . . . 74
8.3 Validity threats . . . 75
8.4 Future work . . . 75
8.4.1 Analysis of automatic entity detection opportunities . . . 76
8.4.2 End-user analysis . . . 76
8.4.3 Implementing the chatbot in a software application . . . 76

Bibliography **77**

Appendix **80**

A Interview protocol **81**

B Software bot taxonomy **87**
B.1 Environment dimensions . . . 87
B.2 Intrinsic dimensions . . . 87
B.3 Interaction dimensions . . . 87

C Pattern accuracy results **88**
C.1 What pattern accuracy . . . 88
C.2 Who pattern accuracy . . . 89
C.3 Why pattern accuracy . . . 90
C.4 Where pattern accuracy . . . 91
C.5 When pattern accuracy . . . 92

D Initial conversation flow **93**
D.1 Full conversation flow . . . 93
D.2 Welcome dialog . . . 93
D.3 Question problem dialog . . . 94
D.4 Expected end-result dialog . . . 94
D.5 Reconstruction scenario dialog . . . 94
D.6 Edit dialog . . . 95
D.7 Rate chatbot dialog . . . 95

E Prototype testsets **96**
E.1 Testset version 0.1 . . . 97

F Prototype example utterances **100**
F.1 Example utterances version 0.1 . . . 100

G Prototype interview protocol **101**

H Developer interview protocol **105**

List of Figures

2.1 Visualization of HiX Modules 4

3.1 Example of the relationships among the aspects of requirements engineering [17]. . 9
3.2 Definition of software bots [27]. 14
3.3 "The relationship between software bot interfaces and software services: (a) software bot with external services, (b) software bot with internal services, and (c) software bot with both internal and external services" [27]. 14
3.4 "Mainstream adoption of new human-computer interface paradigms" [27]. 14
3.5 Difference between software bots and chatbots [27]. 15
3.6 Mock-up of the CORDULA system [12]. 16
3.7 Example of a chatbot with a command-line interaction model. Extracted from https://core.telegram.org/bots 16
3.8 Example of a chatbot with a natural-language interaction model. Extracted from https://algorithmxlab.com/blog/2017/12/19/bring-chatbots-say-5-giant-banks/ . 17
3.9 Chatbots categorized by purposes 18
3.10 High-level taxonomy of software bots [27]. 19
3.11 Environment dimensions [27]. 19
3.12 Intrinsic dimensions [27]. 22
3.13 Interaction dimensions [27]. 28

4.1 The Three Dimensions of Requirements Engineering [35]. 35
4.2 Design science cycle [45]. 36
4.3 Structured vs. unstructured requests 37

6.1 LUIS development cycle [38]. 48
6.2 LUIS request and response [38]. 48
6.3 Bot builder activity [42]. 49
6.4 Bot builder processing stack [42]. 49
6.5 Initial prototype dialogues. 50
6.6 Example Document type entity values. 52
6.7 Precision & Recall per entity version 1. 52
6.8 Second prototype dialogs. 54
6.9 Precision & Recall per entity version 2. 55
6.10 Third prototype dialogues. 57
6.11 Precision & Recall per entity version 3. 58
6.12 Fourth prototype dialogues. 61
6.13 Precision and Recall per entity version 4. 62
6.14 Proto-requirements definition 63
6.15 Recall and precision over versions. 64

7.1 Test set assignment. 65

List of Tables

3.1 Keywords indicating a review type (basic classifier) [30]. 12

7.1 Scoring of chatbot per subject 71

Chapter 1

Introduction

Todays large software systems comprise a vast number of stakeholders. By large software systems one can think of systems such as Microsoft Windows, Apache Webserver, Facebook and other systems that go beyond a single-customer setting and serve a large market of customers.

These stakeholders include customers who pay for the system, users who interact with the system to get their work done, developers who develop the system, and more [9]. However, engaging a large number of software product users, who are beyond an organizations reach, in requirements engineering (RE) by using traditional RE methods has proven to be a challenging task [39]. Traditional market-driven RE approaches elicit feedback through beta-tests, questionnaires and focus groups. This physically limits the amount of feedback and requirements to be gathered by the effort it takes to gather these requirements [17].

More advanced RE approaches applied in market-driven RE provide companies with tools to directly interact with the users of the software product using ad hoc feedback-gathering channels such as forums [15]. However, Snijders et al. state that "these approaches seem to miss the opportunity to continuously involve large, heterogeneous groups of users who express their feedback through a variety of media" [39].

Crowd Requirements Engineering, from now on CrowdRE, aims to gather feedback from a crowd of users or representatives which can be a larger audience than possible with traditional RE techniques [17]. This is achieved by using several unobtrusive automated means of gathering feedback [17]. One can think of gathering feedback through reviews submitted in app stores, feedback from forums and in-app solutions. However, this results in large data sets of feedback which can not be analyzed manually due to time and effort constraints [17][41]. Therefore, a crucial component of CrowdRE is to provide companies with a solution to semi-automatically analyze and transform the feedback to requirements [17].

In 2016 Robert Dale argued that intelligent virtual assistants were the most hyped language technology, Lebeuf et al. agree that bots are quickly becoming the standard for communicating with software services [7][26]. In addition, Shawar et al. state that people want to use natural-language to communicate with computers [37], Zadrozny et al. agrees with this statement by stating that users want "to express their interest, wishes, or queries directly and naturally, by speaking, typing, and pointing" [48]. One can think of the voice-driven digital assistants such as Siri by Apple, Cortana by Microsoft, Alexa by Amazon and the Google Assistant, followed by an enormous range of text-based chatbots [7]. The reason for this sudden rise in popularity is that bots provide developers with a convenient way to generate a UI for interacting with machine-learning algorithms [26]. In addition, major software companies such as Facebook and Microsoft are recognizing and acknowledging the values as bots, Microsoft states that "conversation as a platform" is the OS of the future [26]. Chatbots can therefore prove valuable as a communication

platform between users and developers, providing a feedback medium which can in turn help big software companies gain a better understanding of their users needs.

The goal of this thesis project is to uncover how the feedback of large heterogeneous groups of users can be continuously elicited in the requirements engineering process, by using a chatbot that can be integrated in the software product. This way users can continuously provide feedback using the chatbot on the fly. This feedback is then used as input for the chatbot to elicit requirements from the feedback resulting in proto-requirements. These requirements can be used by the company to improve the quality of their software product. Additionally, these conversations and the resulting feedback can be stored to gain a better understanding of the needs and wishes of the software product users. The research question that will be used in this thesis project goes:

"How can chatbots integrated within a software product prove beneficial in eliciting requirements from user feedback?"

This question will be answered by designing and developing a prototype of a chatbot that can eventually be implemented within a software application. The prototype will then be evaluated by consultants and developers at the case organization, to gather feedback and elicit requirements from their feedback.

This is achieved by executing a case study at ChipSoft where we will develop a prototype of a chatbot that can be implemented in their software product named HiX (Health information Xchange). The consultants and developers at ChipSoft will be asked to provide feedback through the prototype. Finally, the results will be evaluated with the consultants and the requirements generated by the prototype will be discussed with the developers, resulting in a comparison between the situation with and without the chatbot.

In Chapter 2 a detailed description of the case in this research project is given, followed by the literature review in Chapter 3. The literature review positions the research and provides the groundwork for answering the research questions. In Chapter 4 the research method and approach is explained to this end, followed by a detailed description of the conversation flows in Chapter 5. In Chapter 6 the design of the prototype will be described in detail. In Chapter 7 we will delve deeper into the evaluation of the prototype followed by Chapter 8 where the results and conclusions of the research will be discussed.

Chapter 2

Case Description

For the case study, a prototype of a requirements focused chatbot, that can be implemented in the software application HiX by ChipSoft, will be developed. Since HiX is a large and complex software product, it is divided into different so-called "modules" as displayed in Figure 2.1. These modules cover a broad range of fields, e.g. Finance, Patient and Multimedia modules exist. Since it is not possible at this time, and the scope of this thesis project to develop a prototype covering all these modules, the prototype will be developed and prepared for implementation exclusively for the Multimedia/PACS (Picture Archiving and Communication System) module in HiX. This module focuses on displaying, storing and interacting with a range of multimedia documents such as images, videos and audio recordings. This module is an adequate choice, for it is under active development and change. Therefore, a lot of feedback can be gathered in this module.

The consultants at ChipSoft have the role to implement HiX at the customers and tailor the product to their specific wishes. This is accomplished by the consultants creating a prototype of the customers' wishes in a sandbox environment at ChipSoft. This sandbox environment is installed in a testing environment at the given customer allowing the system administrators and other employees to test the given prototype. Finally, if the prototype passes the tests the environment is rolled out into production. In addition, the consultants also handle any customer wishes, provide support and assist the developers in support calls, bug reporting and requesting features. They provide the main communication channel between the developers and the external customers, this makes them the most important internal customer at ChipSoft as seen from a developers point of view.

Due to this fact, and the fact that it is not possible to implement the prototype at actual customers and integrate the prototype in the software application due to time- and security constraints, we choose to develop the chatbot as a stand-alone application and evaluate it likewise. The consultants will be able to interact with the prototype and report bugs through the prototype. The prototype can then use the feedback provided by the consultants to formulate proto-requirements using realistic data.

In the current situation, the consultants can report bugs or handle support calls through a web solution. The communication between the customer and the consultant is monitored, and it provides the possibility to create internal memo's for the communication between consultants and developers. This way the progression of the support call can be monitored from begin to end. However, it does not provide developers with a clear view on what needs to be developed or what the bug exactly is, due to the possible poor formulation of the bugs in question. This often results in not clearly understanding the "why" dimension of the requirement in question. As Yu et al. state, understanding the "why" is an important part of requirements engineering [47]. Therefore, the prototype should focus on uncovering and eliciting the "why" of a requirement through a conversation with the user, in this case the consultants and developers. This will provide ChipSoft

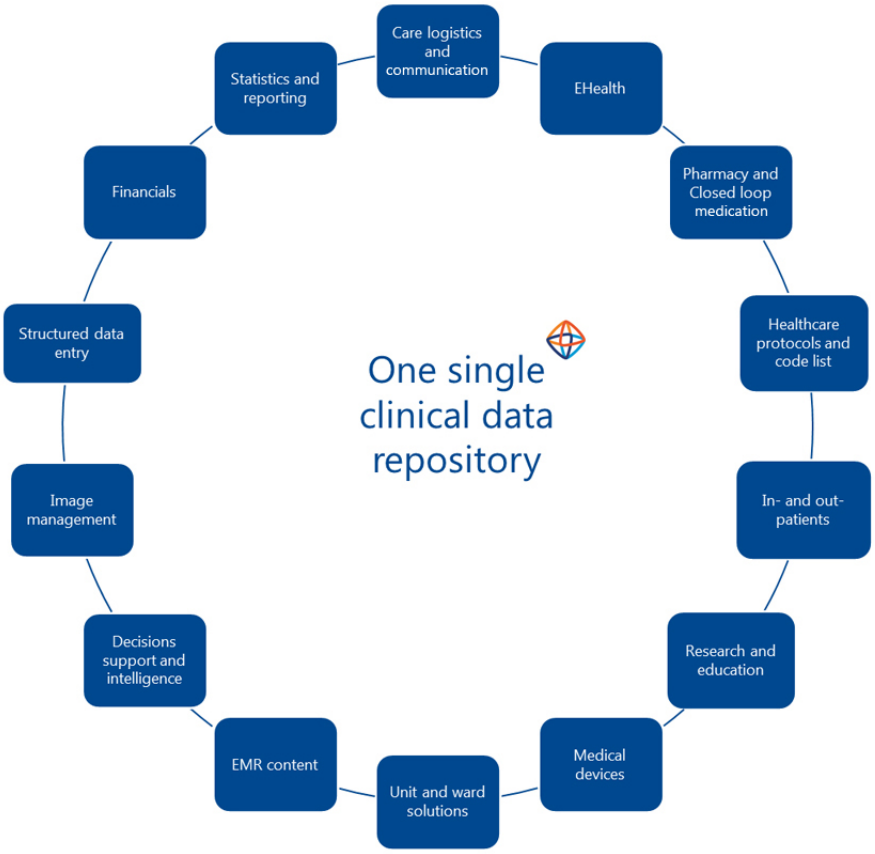


Figure 2.1: Visualization of HiX Modules

with better insights on the rationale of a requirement, than just understanding the "what", offering the possibility to think of other solutions during the requirements engineering process.

The feedback data can be used to make a comparison between the current situation and the situation with the prototype. To do so, we will conduct interviews with the consultants and other stakeholders to identify their requirements and constraints. In addition, the current requirements are compared with the requirements generated by the prototype to evaluate the added value of a chatbot and see which requirements are of higher quality according to the developers. Finally, the usability of a chatbot will be evaluated with the subjects in question.

2.1 Initial prototype requirements

For the prototype to be successful it should satisfy certain concrete requirements. The requirements of the initial design of the chatbot prototype will be listed in this subsection. This list will be complemented by the literature study and results of the semi-structured interviews in Section 3.4. Since the prototype focuses solely on reporting bugs and creating valuable requirements from this feedback, the initial requirements of the prototype are:

- **R1 - Bug problem statement extraction** The prototype should be able to extract a concrete problem statement through conversing in natural-language with the user;

- **R2 - Desired end-result extraction** The prototype should be able to extract the desired end-result through conversing in natural-language with the user;
- **R3 - Classification of a bug** The prototype should be able to classify a bug in terms of location, urgency and users affected;
- **R4 - Reconstruction scenario extraction** The prototype should be able to extract a reconstruction scenario through conversing in natural-language with the user;
- **R5 - Create proto-requirements** The prototype should be able to summarize the bug report and report this back to the user;
- **R6 - Bug summarizing** The prototype should be able to summarize the bug report and report this back to the user;
- **R7 - Synonym knowledge base** The prototype should be able to detect and act upon synonyms of work-related terms e.g. Patientphoto and Image document type.

These requirements describe the prototype on a high-level and highlight the main functionalities. They originate from interviews with the consultants and developers at ChipSoft and discussions with the first supervisor of this thesis project. It focuses on reporting bugs solely due to time constraints, added value and complexity. Requesting new features is of higher complexity than bug reporting due to the higher degrees of freedom the user has in describing a feature. For this reason, a conversation about a bug can be controlled easier. In addition, we expect that there is more added value for conversing through natural-language to report a bug than there is when requesting a feature. The reason for this expectation is that the reconstruction scenario can be elicited more in-depth and follow-up questions might add value.

Chapter 3

Literature Review

To gain a better understanding of the domain area of crowd requirements engineering, from now on CrowdRE, a literature review was performed. The literature review helped us get a better understanding of the current solutions in the domain, and what problems still reside in the domain.

3.1 Requirements Engineering

For software companies it is crucial to ensure that a software product fulfills the needs of the users as accurately as possible. Therefore, the company first needs to acquire knowledge regarding the needs and wishes of users, the so-called *requirements* of the product. According to Paetsch et al. a requirement is used to precisely describe what is to be developed but leaves out how it should be implemented [32]. In contrast, Harwell et al. describe a requirement as a 'thing' that acts as a promise that something must be accomplished, transformed, produced or provided [20]. Finally, Dick et al. define a requirement as "a statement that identifies a product or process operational, functional, or design characteristic or constraint, which is unambiguous, testable or measurable, and necessary for product or process acceptability (by consumers or internal quality assurance guidelines)." [10]. In this research we will not select nor create a definition of requirements as this is beyond the scope of this research.

Requirements engineering is regarded as the process to uncover these requirements by formulating, analyzing and agreeing on what, why and how something should be developed [44]. Dick et al. define requirements engineering as "a subset of systems engineering that is concerned with discovering, developing, tracing, analyzing, qualifying, communicating and managing requirements that define a system of successive level of abstraction." [10]. Requirements engineering can assist in gathering requirements **before** development starts to reduce the risk of having to do costly rework [32]. However, due to the popularity of agile software development, requirements engineering can be seen as a continuous process that can be used before, during and after the software development phase, to uncover fault repairs, environmental adaptation and functionality additions requirements [40][49]. Because this research focuses on gathering requirements from feedback, and therefore after the software development phase, we will approach RE as a continuous process, which is conducted before, during and after the software development phase.

3.1.1 RE activities

In the literature, RE is defined as a process that is split up in different activities or tasks. Thayer and Dorfman split up the RE process into five key activities in their book [43]:

1. Software requirements elicitation - Discovering, reviewing articulating and understanding the needs and wishes of the user;
2. Software requirements analysis - Analyzing the acquired needs and forming them into requirements;
3. Software requirements specification - Specifying the requirements in a requirements document;
4. Software requirements verification - Verify the specified requirements and check if they are in line with standards and useful;
5. Software requirements management - Planning and controlling of the previous defined activities.

While not all research agrees with this set of activities or tasks, requirements elicitation seems to be a returning activity in all the literature. In contrast, van Lamsweerde adds the following activities in their study [44]:

1. Domain analysis - Gathering domain knowledge to gather a greater understanding of the problem to solve;
2. Evaluation and agreement - Aims on making informed decisions of issues arised during the elicitation activity;
3. Requirements documentation - Regards the creation of a requirements document containing all the requirements;
4. Requirements evolution - Activity intersects with change management and configuration management in software engineering.

This research focuses on a combination of the elicitation, analysis and specification activities. Where a chatbot will serve as the elicitation mechanism handling the analysis and specification using feedback. Therefore, we will not give an in-depth description of the remaining activities as it is beyond the scope of this research.

3.1.2 Requirements elicitation

The requirements elicitation activity aims at uncovering the requirements that will help shaping the system-to-be. In this activity it is crucial to gather information about the application domain, business needs, system constraints, stakeholders and the problem itself to better understand what should and should not be developed for the system-to-be [32][44]. To achieve this, a number of techniques can be used, van Lamsweerde splits up these techniques in *artefact-driven* and *stakeholder-driven* techniques [44]. Since this research focuses on creating requirements from feedback, thus stakeholder-driven, we will not delve deeper into artefact-driven techniques.

Stakeholder-driven elicitation techniques rely on interaction with stakeholders to obtain relevant information about the organization, domain and the problems residing in the system-as-is [44]. van Lamsweerde propose the following techniques [44]:

1. Interviews - Structured and unstructured interviews can be conducted among stakeholders;
2. Group sessions - A series of group workshops can be organized to elicit feedback from larger groups of stakeholders;
3. Observation - Observing how stakeholders use the software to uncover requirements.

In addition, Paetsch et al. add the following technique to the list [32]:

1. Brainstorming - Organizing brainstorming session to gather creative ideas focused on evolving the software product.

While these techniques are useful for a small amount of stakeholders, the amount of interaction with the stakeholders is inefficient when trying to elicit requirements from a large number of end-users. This is because there is a physical limit in the amount of requirements that can be gathered by the effort it takes to gather these requirements [49]. Therefore, Zowghi et al. advice investigating methods that involve direct interaction with stakeholders using new technologies including web and agent based architectures [49].

3.1.3 Requirements analysis

The requirements analysis phase consists of checking the necessity, consistency, completeness and feasibility of the elicited requirements. This is in the majority of the cases a manual task to be executed by a requirements analyst. The task commonly consists of JAD (Joint Application Development) sessions, prioritization and modeling [32][44]. If a conflict is uncovered in the set of requirements, they can be resolved by a prioritization negotiation with the stakeholders to uncover more critical requirements. The final result of this phase is to compromise a set of requirements that is agreed on by the stakeholders [32].

3.1.4 Requirements specification

The purpose of documenting and specifying requirements, is that it is easier to communicate documented requirements between stakeholders and developers [32]. This phase consists of documenting the set of requirements that is agreed upon, in the requirements analysis phase, in the so-called requirements document [32][44]. The requirements document can be the baseline for evaluating products and processes and can be useful for change control. As Paetsch et al. state "A good requirements document is unambiguous, complete, correct, understandable, consistent, concise, and feasible." [32].

3.2 CrowdRE

As this research consists of a case-study at a company developing a software product for the healthcare industry, we will limit us to the approach of market-driven RE. This is because the product goes beyond a single-customer setting and this type of approach enables serving a large market of customers [17]. Todays large software systems comprise a vast number of stakeholders. These stakeholders include customers who pay for the system, users who interact with the system to get their work done, developers who develop the system, and more [17]. However, engaging a large number of software product users, who are beyond an organization's reach, in requirements engineering (RE) by using traditional market-driven RE methods has proven to be a challenging task [16][39]. In traditional market-driven RE approaches, feedback is elicited through beta-tests, questionnaires and focus groups. This physically limits the amount of feedback and requirements to be gathered by the effort it takes to gather these requirements. However, there are more advanced RE solutions that use multi-modal feedback gathering approaches [17][16][39] leaving the company with the possibility to directly communicate with their end-users. As observed by Snijders et al., "these approaches seem to miss the opportunity to continuously involve large, heterogeneous groups of users who express their feedback through a variety of media." [39].

In CrowdRE the feedback comes from a crowd of users or their representatives which is a much larger audience than previously [17]. This feedback can be gathered using several unobtrusive automated means, for example, an integrated chatbot in the software product [17]. A big challenge remains in motivating the crowd to provide high-quality feedback, the authors state that digital-motivation techniques should be adaptive to the context and adaptable by the crowd, but in such a way that it does not affect the quality of the feedback [17]. A chatbot could be such a

digital-motivation technique as it is adaptive and can easily be adopted by the crowd. In addition, it allows the user to further specify their feedback by using natural language in a conversation.

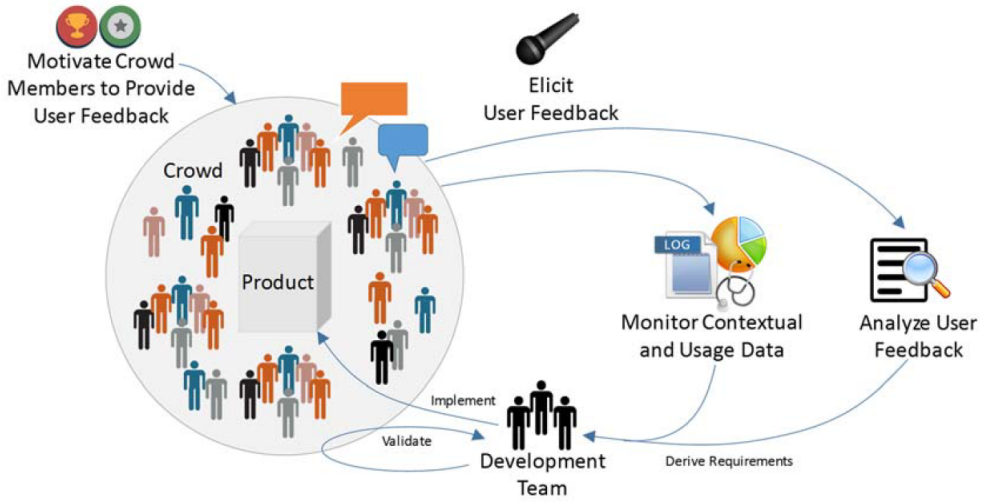


Figure 3.1: Example of the relationships among the aspects of requirements engineering [17].

A CrowdRE solution should strive to gather a continuous stream of feedback from as many crowd members as possible to identify and communicate the needs of these crowd members [17]. The feedback and information is called user feedback where the crowd member (or user) is considered as the direct sender of this feedback. The authors divide this entire process in five different key activities [17]:

1. Motivating crowd members
2. Eliciting user feedback
3. Analyzing user feedback
4. Monitoring contextual and usage data

3.2.1 Motivating crowd members

The motivation activity is a very hard one as it focuses on motivating users to provide a continuous stream of user feedback. A crowd member can be motivated in two different ways: 1) intrinsic and 2) extrinsic. Intrinsic is when the crowd member is genuinely interested in the software product and willing to commit time in the software evolution [17]. Extrinsic is when the feedback results from external interventions and incentives such as monetary rewards and non-monetary rewards such as social recognition [17]. As stated by Hosseini et al. "Intrinsic motivations are typically the stronger of the two" [22]. The challenge at hand is to create interest in such a way amongst crowd members that they are intrinsically motivated to provide continuous feedback. However, the challenge in this activity is that motivating the crowd has been proven to be extremely hard [31]. As stated by Snijders et al. gamification and persuasive technology, such as rewarding crowd members for providing feedback, can be used to boost the interest of crowd members in providing a continuous stream of valuable feedback [17][39][29].This research will focus on getting crowd members intrinsically motivated as there will be no reward for using the chatbot to provide feedback.

3.2.2 Eliciting user feedback

The next key activity involves the process of eliciting feedback from the crowd members and can be seen as the most interesting key activity in this research. Deepa et al. state that "existing methods for requirements elicitation require intensive interactions with the stakeholders, for example, through face-to-face meetings, interviews, brainstorming sessions, and focus groups" [9].

There are two ways in which feedback is gathered, pull feedback is when the feedback is gathered from the crowd through a specific request made by the company, push feedback is when the crowd independently provides feedback [17]. In this research we mostly focus on the push type of feedback as we provide the crowd with a chatbot which they can independently use to provide feedback through. An additional source of feedback can come from monitoring data, this means that feedback can be enriched by monitoring the end-users behavior [17][41]. This extra information helps the developer in providing them with a deeper understanding of the particular needs and problems of the crowd [41]. This research will focus on gathering linguistic feedback by providing crowd members the possibility to communicate their feedback through natural language. Crowd members report on a lot of different things ranging from bugs, extension ideas and new product ideas [17][31]. Since the eliciting of requirements was an activity formerly executed as a design time activity it is currently shifting more and more to a run-time activity, actively involving the crowd members via the use of user feedback [23][2]. This feedback is given through a number of feedback channels such as forums, app-store reviews and social media [17]. While this is a step in the right direction, the crowd members need to go out of their way to provide feedback posing them to a potential hurdle for providing feedback [17]. This could mean that there should be an easy way for crowd members to provide feedback on the fly rather than having to take an extra step to provide feedback. The combination of these new feedback elicitation methods and the current more traditional elicitation methods can lead to a very powerful combination, that leaves the developer with a greater understanding of the wishes and needs of their crowd members [17].

3.2.3 Analyzing user feedback

To actually leave the developers with a greater understanding of the crowd members, the user feedback needs to be analyzed [17]. By providing the crowd members with a way to provide feedback on the fly, the amount of feedback may also increase. This could lead to the point where it is no longer possible to analyze the feedback manually [17][41]. Therefore, an automated way of analyzing the feedback would be almost vital to benefit from the big set of user feedback [17]. The most obvious solution is to use linguistic analysis techniques such as text-mining to automatically analyze the feedback [17]. This would classify the feedback in different categories such as "bug reports" and "feature requests" [17]. However, a challenge is to reduce the risk of so-called "misuse" which could lead to accidentally extracting personal information from crowd members [31]. Therefore, it is very important to provide crowd members with a way to set their privacy preferences. In this way both the company and the crowd members can benefit from sending and receiving feedback.

3.2.4 Monitoring contextual and usage data

Another step in the process is the monitoring of context and usage data. In the future there can be new ways to gather user feedback, for example gathering usage data through sensors in internet of things devices or through the use of usage-mining techniques [17][16]. The combination of both the text-mining in the analysis phase and the usage-mining techniques used in this phase can provide the company with even more valuable information about the crowd. In addition, combining usage data with feedback can be very valuable when two opposing opinions arise to make a decision [16]. Finally, usage data can also be monitored in software products, which can also lead to unveiling new and formerly unknown requirements without the involvement of the user itself [17].

3.3 User feedback

For software companies it has become increasingly important to involve their users in the software development process [3][24][34][5][18]. Bragge et al. state that receiving innovative end-user feedback is an essential part for acquiring new development ideas [5]. In the long-term, actively involving users in the software development process can have a positive impact on the success of the software product [49]. According to Damodaran, actively involving users in the software development process can yield the following benefits [8]:

1. Improved quality of the software product due to more accurate user requirements;
2. Avoidance of costly features that the user do not want and will not use;
3. Improved level of acceptance of the software product;
4. Greater understanding of the software product by the user;
5. Increased participation in decision making.

However, it often proves difficult for software development companies to actively involve users in their software development process [18][24]. This is because, for most of the software development companies the customer is prioritized over the end-user. In addition, the developers are often unable to interact with or communicate with the end-users, resulting in the requirements being communicated through marketing. Finally, because of the popularity of agile software development, and the fact they use relatively short development cycles, there is no time for involving the users in every iteration [18][24].

Panichella et al. state that user feedback can contain usage scenarios, bug reports and feature requests [34]. However, in popular applications this will lead to a large set of feedback. As feedback varies in quality and tends to be unstructured identifying valuable user feedback proves to be a challenging task [34].

Due to the increased popularity of mobile applications, application distribution platforms or app stores such as the Google Play or Apple AppStore have become increasingly popular [13]. These app stores allow to submit feedback, which is particularly interesting from the software and requirements engineering perspective, in the form of *reviews* and so-called star-ratings [13][34][33]. However, Pagano et al. state that the real potential and impact on requirements engineering are not yet well understood [33]. These reviews consist of rating the application with a number of stars and providing a review message. They are completely public and can be seen by other people and the developers of the application [33]. Reviews serve as a communication channel between developers and users where the users can provide the developers with relevant information about the application which can help software development companies improve their product quality [33][34]. The reviews provided in the app stores seem to differ from other online stores in two ways [13]:

1. The reviews seem to be shorter in length;
2. An app can have multiple releases. Therefore, reviews can be specifically aimed at a certain version and can vary over time.

These reviews provide a strong medium for eliciting feedback and communication between developers and users. However, due to the size of the set of reviews, it requires a large time investment to manually analyze, process and extract valuable information from these reviews [13][34]. Additionally, the user has to take an extra step to provide the feedback, possibly missing some feedback from

Review type	Keywords
Bug reports	bug, fix, problem, issue, defect, crash, solve
Feature requests	add, please, could, would, hope, improve, miss, need, prefer, request, should, suggest, want, wish
User experiences	help, support, assist, when, situation
Ratings	Great, good, nice, very, cool, love, hate, bad, worst

Table 3.1: Keywords indicating a review type (basic classifier) [30].

3.3.1 Feedback classification

Due to the fact that it is time-consuming to manually analyze the large amount of feedback made available through app store reviews it is beneficial to automatically classify reviews. Maalej et al. state that reviews can effectively be categorized in four types [30]: 1) Bug reports, 2) Feature requests, 3) User experiences, 4) Ratings. Bug reports describe problems with the app that should be fixed, think of crash reports, performance issues and erroneous behavior. In feature requests the users suggest or asks for missing content, functionalities and features and share ideas on how the software product can be improved in the future. User experience describes the experience and perception of the user with the software product in certain situations and can be seen as the documentation of the product, its requirements and features. Finally, the ratings are less informative for developers and basically just reflect the overall perception of the software product [31][33].

However, it is a challenging task to automatically understand what a user means in a certain review. Therefore, Panichella et al. state that "a deep analysis of the sentence structure needs to be exploited to determine the intention of a review' [34]. In addition, the tense of a certain review can also change the meaning of the review [34].

They have used review metadata such as the star rating and tense of the sentence in combination with text classification, natural language processing and sentiment analysis techniques to automatically classify the reviews in one of the four types previously mentioned [30]. In addition, they have introduced various classification techniques to automatically classify reviews [30].

Basic Classifier: String Matching

The first, and most basic classifying technique mentioned is to check whether it contains a certain, manually configured, keyword. These keywords are added to a list of keywords specific for a certain type of review previously mentioned and check if the review contains one of the keywords on this list. For this, techniques such as regular expressions, string matching and SQL queries can be used [30]. Maalej et al. have identified a list of keywords indicating a review type, displayed in Table 3.1.

Document Classification: Bag of Words

Document classification is a technique in information science where a document is assigned to a certain class. A basic example is the classification made between "spam" or "no spam" e-mails. In the case of reviews, a single complete (title and text) review is regarded to as a document. The basic form of document classification is called bag of words (BOW), where the classifier creates a dictionary from all the terms in the corpus of all reviews and calculates if and how often a term is present in a certain review [31]. The advantage of this technique is that machine learning algorithms can be used to extend this technique based on review metadata. In addition, it does not require the developers to manually maintain and set up a list of keywords and patterns of keywords can be used to predict the type of a review [31].

Natural Language Processing: Text Preprocessing

To increase the classification accuracy some pre-processing of the reviews, such as stopword removal, stemming, lemmatization, tense detection and bigrams, is required. Maalej et al. suggest using common natural language processing techniques for this task [30].

Stopwords are referred to as "common English words such as the, am, their" that do not interfere or add something informative to the review. By removing this "noise" the influence of more informative terms such as "bug" or "add" can be increased. This will directly improve the accuracy of document classifiers [30]. However, they mention that some stop words such as "should", "but", "before", "while", etc. can possibly prove to be relevant in classification [30].

Lemmatization is the process that reduces different forms of a word to their basic lemma, so that it can be analyzed and handled as a single item [30]. Additionally, *stemming* reduces each word to its basic form by simply removing the post-fix. The difference is that lemmatization uses dictionaries and takes the linguistic context of a word in consideration where stemming does not. Both of these techniques can prove to be useful by reducing the amount of different keywords to their basic forms and hereby, increasing the count of these keywords. This allows the classifier to learn faster and more efficiently and allows the use of sequences of words that co-occur more often than by chance [30].

Review Metadata: Rating and Length

The reviews contain metadata that can be collected such as the star rating, length and the submission time. These star ratings are, most commonly, a numeric value between 1 and 5 where 1 is the lowest score and 5 the highest possible. The star rating could help in classifying a review, as for example, a bug report will most likely have a low star rating. Maalej et al. state that user experience reviews are most likely to be found in positive reviews with 4 or 5 star ratings [30]. The length of certain reviews can be used as an indicator of a reviews' type. A longer review might be more valuable and informative, as it indicates a report on an experience or maybe a bug [30]. Finally, Maalej et al. use the tense of verbs to classify reviews, they state that a past tense might be used for reporting where a future tense might indicate a feature request [30].

Sentiment Analysis: Sentiment Scores

Reviews in apps stores might indicate a user's sentiment, this sentiment can be used in the classification of the reviews. A negative sentiment might indicate a bug report where a positive sentiment might indicate a user experience. These sentiments, might be extracted from the reviews and used by the classifier to train itself in identifying the type of reviews more accurately [30].

Supervised Learning: Binary vs. Multiclass Classifiers

Finally, a review can be of more than one type e.g. a complaint or negative rating and a bug report. This might prove problematic as the classifier will not be able to classify the given review. However, supervised machine learning algorithms might prove a solution to this. These algorithms will calculate the probability for each factor to decide whether the review is of a type or not. In contrast, a multi class classification creates the possibility to assign a review to multiple types [30].

3.4 Chatbots

Chatbots are computer programs that interact with human-beings using natural language through a text interface [37]. More in-depth Lebeuf et al. specify **chatbots** as a **software bot** and defines a software bot as "an interface that connects users to services" as can be seen in Figure 3.2 [27].

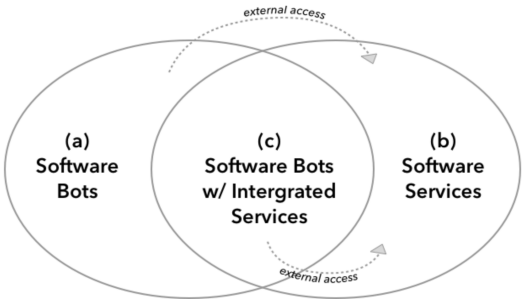


Figure 3.2: Definition of software bots [27].

As visualized in Figure 3.3 services can then internalized and/or accessed externally. In addition, the bot should provide additional value, for example in the form of interaction style or automation on top of the software service's basic capabilities [27].

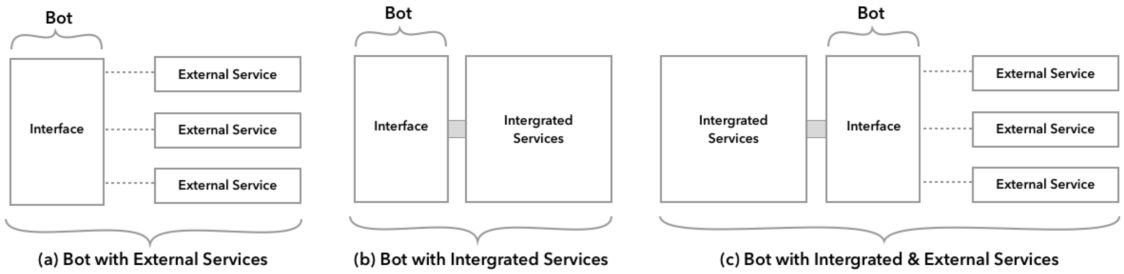


Figure 3.3: "The relationship between software bot interfaces and software services: (a) software bot with external services, (b) software bot with internal services, and (c) software bot with both internal and external services" [27].

Many near-synonyms exist for chatbots such as dialogue systems, interactive conversational agents, virtual agents and chatterbots [36]. While chatbots were first developed to fool humans that they were talking with an actual human being. However, today there are a lot of new applications for chatbots that focus on supporting humans in their daily tasks as opposed to fooling humans that they are talking to an actual human being [37][7][36].

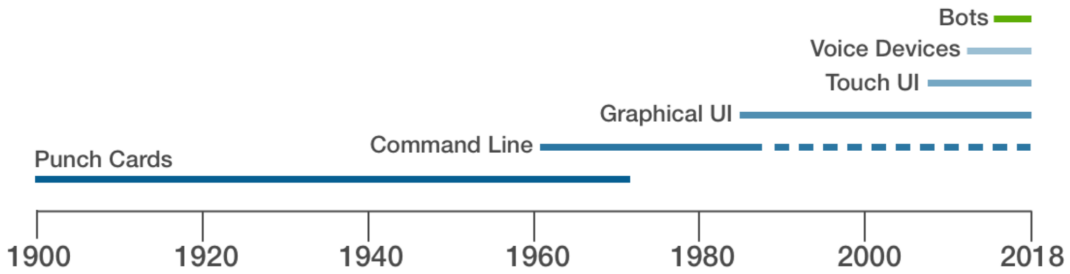


Figure 3.4: "Mainstream adoption of new human-computer interface paradigms" [27].

These chatbots can be used for a large range of applications such as technical support, education, language learning tools and entertainment [37]. Dale also mentioned that the MIT Technology Review has listed conversational interfaces, thus chatbots, as one of the ten breakthrough technologies in 2016. He also reports that "Uber's Chris Messina wrote an influential blog piece that 2016 is the year of conversational commerce and Satya Nadella announced that chatbots were the

next big thing in the same year" [7]. However, these were only mere speculations and hype, the fact remains that interaction with technology using natural language is become more and more significant, popular and feasible for many purposes [7]. In addition Lebeuf shows the mainstream adoption of human-computer interface paradigms in Figure 3.4 which supports the statements made previously and showing the recent popularity and interest in bots [27].

Lebeuf states that there is a significant difference between a chatbot and a bot as displayed in Figure 3.5. She states that a *chatbot is always a bot* but a *bot is not always a chatbot* [27]. She states that chatbots are distinguished by their ability to communicate with users by using natural language [27]. Lebeuf describes a chatbot as *"any software bot with a conversational interface"* [27].

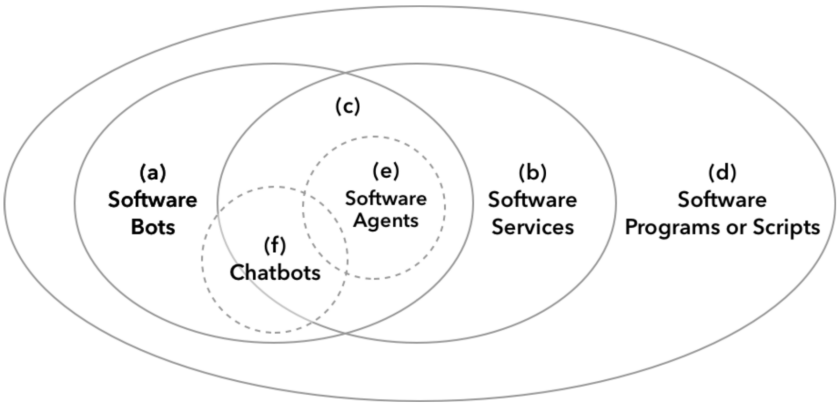


Figure 3.5: Difference between software bots and chatbots [27].

There are different ways to characterize bots, Lebeuf et al. states that a way to characterize bots is by the way the user interacts with a chatbot. They state that characteristics of chatbots are [26]: 1) Interaction model, 2) Initiation, 3) Intelligence, 4) Purpose.

Friesen et al. have developed a system named CORDULA (Compensation Of Requirements Descriptions Using Linguistic Analysis), as displayed in Figure 3.6, which is a system that uses chatbot technology to support requirements elicitation [12]. During the development of this system they faced four challenges: 1) missing information, 2) stalemate situations, 3) contradictory information, 4) known but not solvable problems. They solved these challenges by relying on end-user interaction with the system and not rely on the system to make up requirements [12]. While this system is a good first step, it does not work with user feedback and relies on the end-user in certain situations. These end-users have no technical background and therefore should be supported by the chatbot [12]. Another issue residing in this research is that it is not clear to what extent they have actually implemented their ideas or if it is just a concept.

3.4.1 Interaction model

Some bots exclusively support a domain-specific language, this is quite similar to interacting with a command-line interface, meaning that the bot only reacts to a list of defined commands, as illustrated in Figure 3.7. As can be seen the bot has a list of pre-defined commands to execute certain actions.

Other bots interact with users in natural-language, as displayed in Figure 3.8 and parses the valuable information from natural-language conversations [26]. Since this thesis project focuses on gathering feedback from users, who do not have domain-specific knowledge, we will focus on

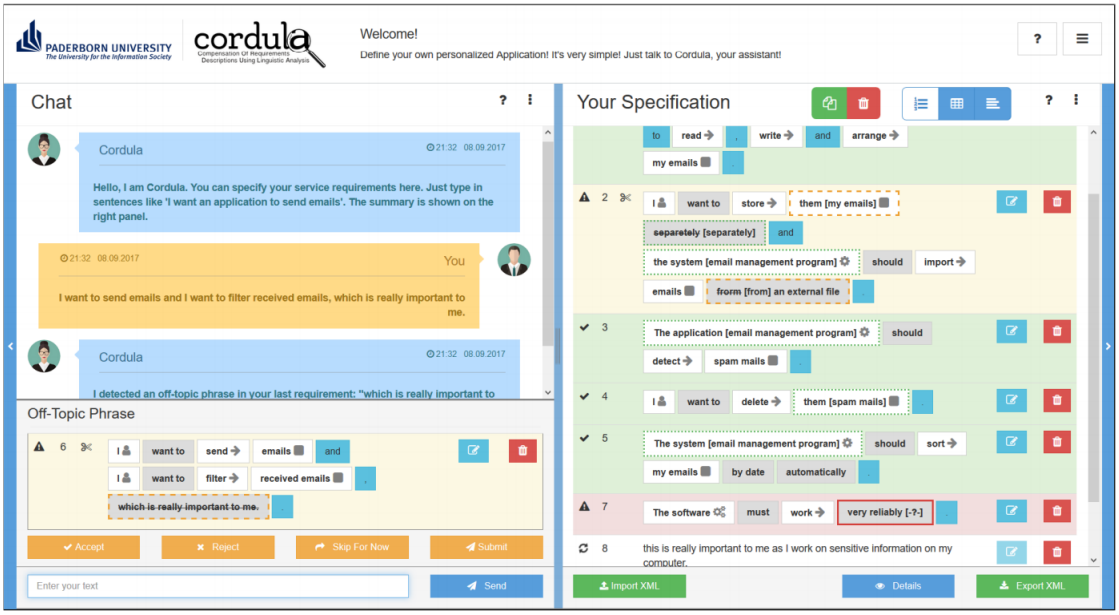


Figure 3.6: Mock-up of the CORDULA system [12].

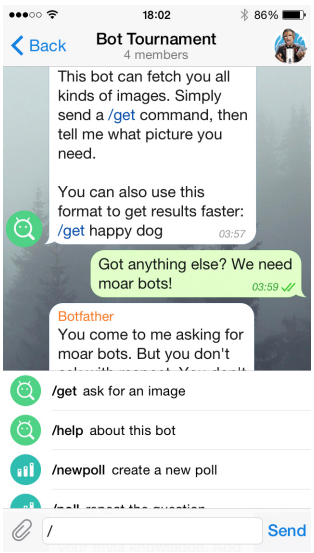


Figure 3.7: Example of a chatbot with a command-line interaction model. Extracted from https://core.telegram.org/bots

gathering feedback using a chatbot that supports natural-language parsing. This gives users more freedom when providing feedback, but makes the parsing more complex and error-prone. More error-prone because since there is no default format each user can communicate requirements in its own way, therefore, the chatbot needs adapt to the situation to effectively elicit the requirements from the conversation.

3.4.2 Initiation

Initiation describes the way the chatbot is initiated for interaction. Lebeuf et al. describe that a bot can support one of to initiation approaches [26]: 1) Pull-based approach, 2) Push-based

Figure 3.8: Example of a chatbot with a natural-language interaction model. Extracted from https://algorithmxlab.com/blog/2017/12/19/bring-chatbots-say-5-giant-banks/

approach. A chatbot that uses a pull-based approach lets the user initiate interaction with the chatbot, this can be achieved by having an initiation message such as "Hey Google", but can also be a simple greeting. A push-based approach is when the chatbot initiates the interaction according to some system or user context [26]. Due to the fact that gathering feedback is done through unobtrusive automated methods [17], the prototype should use a pull-based approach. This leaves the user the choose whether or not to provide feedback.

3.4.3 Intelligence

Lebeuf et al. split up the intelligence of chatbots in three different parts: 1) Adaptation, 2) Reasoning 3) Autonomy. Adaption is described as the degree in which the chatbot is context-aware and is able to use the context to change the way they interact with the users [26]. For this thesis project this is an important factor, the chatbot should be able to adapt to the users input to further classify feedback or ask follow-up questions when it is uncertain.

Reasoning is described as the bots ability to reason, some use simple conversation flows or logic rules where others use more advanced AI to drive their conversations [26]. A first setup of the prototype can be developed by using simple logic rules and a conversation flow, however, it is desirable to implement AI in the chatbot to make it smarter and more adaptive.

Lebeuf et al. describe that some bots are entirely autonomous where some others rely on human input or have a mixed approach [26]. The chatbot proposed in this thesis project should be completely autonomous, since it is very labor-intensive if human-input is needed to gather large amounts of feedback.

3.4.4 Purpose

Lebeuf et al. propose five different purposes for chatbots [26]:

- **Generalist bots** - Similar to Siri and Cortana supporting users in a range of simple tasks

- **Transactional bots** - These bots can automatically execute transactions with external systems
- **Productivity bots** - Improve users or team productivity by automating tedious tasks
- **Informational bots** - Fetch information for users
- **Collaboration bots** - Communicate, coordinate and collaborate

However, this list is incomplete without the mentioning of social chatbots, as stated earlier chatbots were first developed to fool a human in thinking that they are talking to a human instead of a computer. Therefore, we add the purpose **Social bots** to the list which represents the bots that are used for having simple conversations about different topics. Finally, the chatbot proposed in this thesis project is not compatible with one of the purposes described by Lebeuf et al. However, it can be seen as a combination between a productivity and informational bot, as it improves the teams productivity and can at the same time assist the users and provide them with information.

We have made a selection of 38 chatbots, of some of the most impactful chatbots through time and categorized them, in Figure 3.9 in the purposes as stated by Lebeuf et al. [46][11][14].

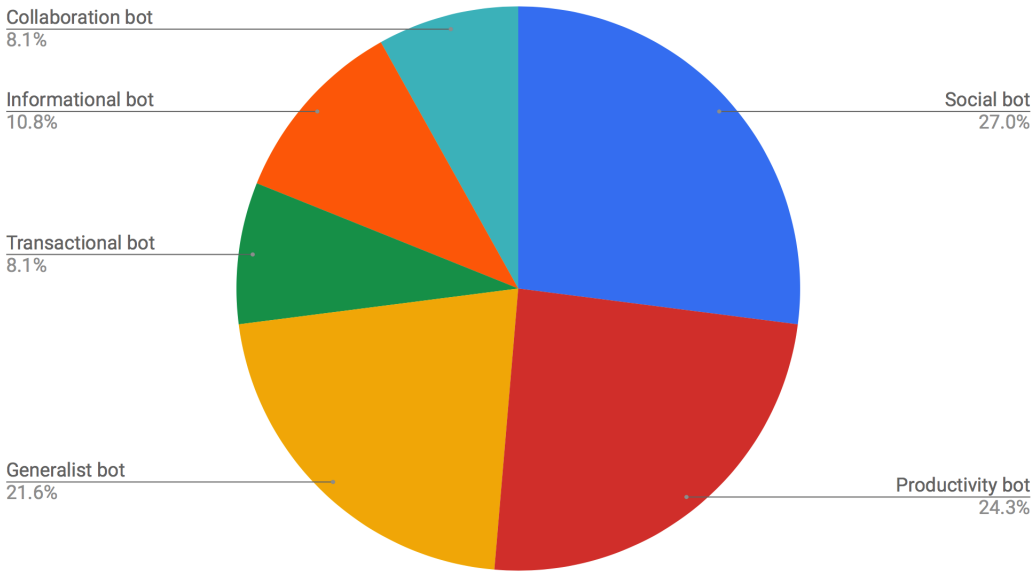


Figure 3.9: Chatbots categorized by purposes

In addition, Serban et al. describe that dialogue systems can be divided into goal-driven systems, such as support services and non-goal driven systems such as computer game characters [36].

While these purposes provide us with insights in the different categories of chatbots and help define the purpose of the chatbot prototype more in-depth. However, classification is needed to further classify the prototype. Moreover, the prototype can not be placed in a single purpose as proposed by Lebeuf et al. Therefore we will classify the prototype using the taxonomy provided by Lebeuf et al. as displayed in Figure 3.10

Lebeuf et al. state that bots can be categorized by using three dimensions that describe the first level of the taxonomy [27]:

1. The environment dimensions - Describes the environment the bot operates in;
2. The intrinsic dimensions - Describes the way the bot is built;

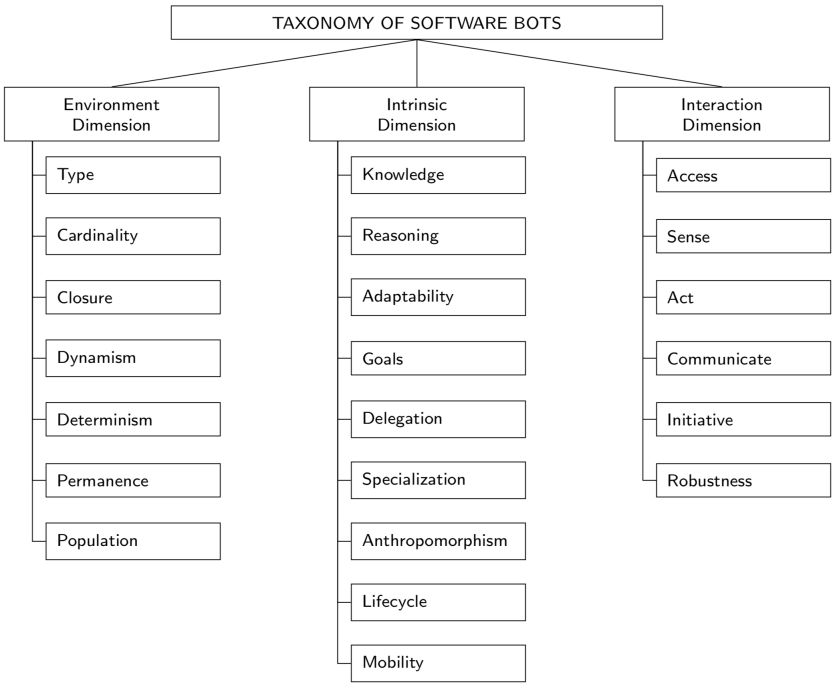


Figure 3.10: High-level taxonomy of software bots [27].

3. The interaction dimensions - The way the bot interacts with the environment.

We will be going through each dimension one-by-one handling each facet and categorize the prototype. In addition, the taxonomy can be used to acquire the requirements that can be used in the development of the chatbot prototype.

3.4.5 Environment Dimension

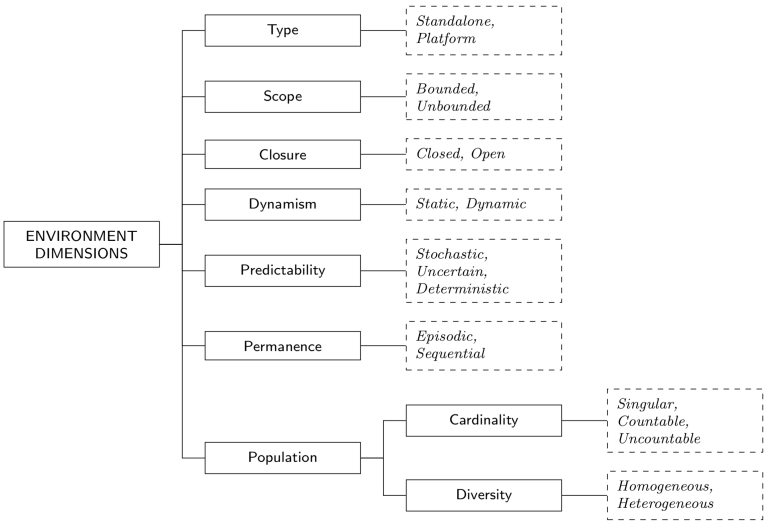


Figure 3.11: Environment dimensions [27].

Lebeuf et al. state that the environment of the bot and the way the bot interacts with this environment has a certain effect on the bot [27]. As shown in Figure 3.11 this dimension has seven facets [27].

Type

In this facet Lebeuf et al. describe two options **Standalone** and **Platform**, these are exclusive and only one can be chosen [27]. Standalone is when the bot is not tied or restricted to a specific platform where platform is described as "integrated" into a platform.

As the prototype will be "integrated" into a platform, a software product. Therefore, we can say that the prototypes' type should be **Platform**.

Scope

The bots scope defines the size of the environment and Lebeuf et al. provides us with two exclusive scope states [27]:

1. Bounded - The bot is limited in how far it can travel;

2. Unbounded - The bot is not limited in how far it can travel.

In our situation, due to the fact that the bot is integrated in a software product, the bot can not operate outside of this product. Therefore, we could say that the scope of the prototype should be **Bounded**.

Closure

The closure facet is concerned about who is able to access the environment and offers us two exclusive states [27]:

1. Closed - Access to the environment is limited by e.g. login;

2. Open - Access to the environment is open.

This facet depends on the way the software product is developed, some software products may be freely accessible and others might require some kind of login. Looking at our case study, in which the software product requires a user to login, we can say that the prototypes' closure will be **Closed**.

Dynamism

The dynamism of a bot is described as the degree to which the bot's environment is capable of being changed by outside forces and provide us with two exclusive states [27]:

1. Static - The bot's environment can not be changed;

2. Dynamic - The bot's environment is susceptible to change;

As a software product is constantly susceptible to change and evolves over time, the environment of the prototype can, and will constantly change. Therefore, we could say that the prototype should be **Dynamic**.

Predictability

The predictability of the bot is described as "the degree to which, given the same conditions, the outcome of the bots actions can be predicted" and gives us a number of options [27]:

1. Stochastic - The actions performed by the bot are random and cannot be predicted;
2. Uncertain - The actions performed by the bot can be partially predicted;
3. Deterministic - The actions of the bot can be fully predicted.

In the case study the actions performed by the prototype can be partially predicted and is therefore **Uncertain**. This is because the prototype will partially depend on the training it gets to define how accurate it can detect and react to dialogues.

Permanence

The permanence is described as how long the effects on changes from actions performed by the bot in it's environment persist [27]. It leaves us with two boolean facets [27]:

1. Episodic - The changes resulting from actions by the bot are not persistent on the environment and have no effect on the future state of the environment;
2. Sequential - The changes resulting from actions by the bot are persistent on the environment and have persistent effect on the future state of the environment.

Since the prototype will be used for providing feedback and will only use the feedback to elicit proto-requirements, the protoype will be **Sequential**. The actions performed by the prototype will not have any persistent effects on the environment.

Population

Lebeuf et al. describe the population as the active entities within the bot's environment, making a distinction between "passive" and "active" objects. Passive objects can only be manipulated and not be changed where active objects can change and can be other bots, humans and systems [27]. This facet can be described by two sub-facets [27]:

1. Cardinality - Describes the size of the population;
2. Diversity - Describes the composition of the environment.

Cardinality

Lebeuf et al. provide three options for the cardinality of the environment [27]:

1. Singular - The bot is the only member of the population;
2. Countable - The population can be reasonably counted;
3. Uncountable - The population can not be reasonably counted;

As the prototype will be part of a software product with a measurable population, the cardinality should be **Countable**. Each user is registered in the system and it is not an open-source or social media platform, resulting in a limited and measurable number of users.

Diversity

The diversity can be described by choosing one of two options [27]:

1. Homogeneous - All members of the populations are the same type;
2. Heterogeneity - The population is not homogeneous.

The prototype can be categorized in the **Heterogeneity** option as it will most likely contain objects of different types.

3.4.6 Intrinsic Dimension

Lebeuf et al. state that the intrinsic dimension describes the internal properties of the bot itself, which are a result of the bot's developers with a clear focus on the externally observable intrinsic properties [27]. The dimensions consists of 7 facets with 24 sub-facets as displayed in Figure 3.12.

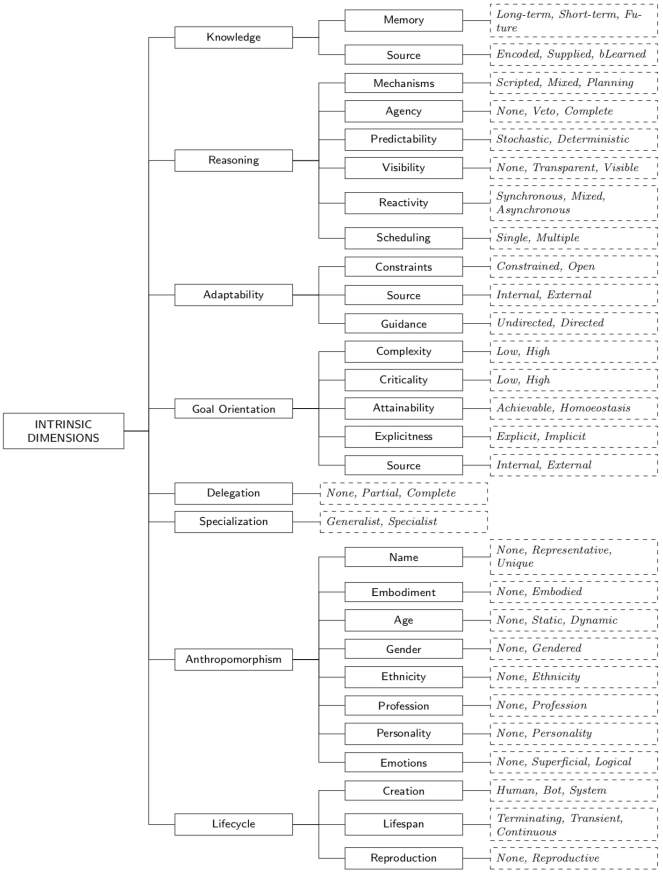


Figure 3.12: Intrinsic dimensions [27].

Knowledge

A bots knowledge is described as what the bot knows and understands. Since knowledge is a very high-level concept, it is broken down into the following sub-facets: memory and source [27].

Memory

Lebeuf et al. state that a bots memory describes its ability to both store and access its knowledge [27]. They offer three boolean values:

1. Long-term - The bot remembers what happened before;

2. Short-term - The bot is able to temporarily, the bot understand where it is, when it is and who it's talking to;

3. Future - The bot is able to store and access predictions of future events.

The interviewees state that the long-term storing and future isn't really necessary in the first place and the focus should be on short-term memory. However, in the future the storing of conversations could prove to be valuable. Therefore, we think that the **Long-term and Short-term** values should both be true for the prototype.

Source

The knowledge source describes where the bots knowledge originates from [27]. Three boolean values are offered:

1. Encoded - The bot's knowledge should be encoded;

2. Supplied - The bot's knowledge is provided by someone or something in its environment;

3. Learned - The bot's knowledge is inferred from it's environment

The interviewees state that the bots' knowledge should partly be encoded to avoid leaking patient data. In addition the bots knowledge will be supplied by the environment and inferred from the environment. Therefore **all values should be true**.

Reasoning

A bots reasoning describes its capacity to apply logic to achieve its goals. Since reasoning is a very high-level concept, the reasoning dimension is broken down into six sub-facets [27].

Mechanisms

The mechanism of a bot describes the way it processes inputs and/or generates outputs in order to realize its goals [27]. The possible reasoning mechanisms range from scripted to planning:

1. Scripted - The bot only responds to predefined stimuli;

2. Mixed - The bot uses a combination of planning and scripted reasoning mechanisms;

3. Planning - The bot has no predefined script mapping inputs to outputs.

Since the prototype in this study should be able to converse in natural-language but should also be able to detect certain keywords, the mechanism should be **mixed**.

Agency

A bots agency describes its ability to perform the tasks it requires to achieve its goals without interference [27]. The possible agency values range from none to complete:

1. None - The bot requires permission of an external party when performing actions;

2. Veto - The bot has the ability to carry out the tasks required to realize its goals, however, an external party can veto the bots actions.;

3. Complete - The bot does not require permission to carry out the tasks required.

The prototype should always be able to create a software request or proto-requirement from the given input. Therefore, the prototype should have the **complete** value. In addition, the prototype will mostly gather data and should not execute any actions that are irreversible.

Predictability

The predictability describes the degree to which the bots output can be predicted given the same conditions [27]. The possible predictability values range from stochastic to deterministic:

1. Stochastic - The bots results appear to be random;
2. Mixed - The bot is sometimes stochastic and sometimes deterministic;
3. Complete - The bot is predictable given the same conditions.

The prototype will rely on the user's input and the way the natural language is processed. For this reason the output and results may sometimes differ from the expected result and therefore, the prototype should have the **mixed** value.

Visibility

The visibility describes the degree to which the bots makes it's decisions or actions visible to others [27]. The possible visibility values range from none to transparent:

1. None - All decisions and actions are hidden;
2. Transparent - Decisions and actions leave visible traces;
3. Visible - Bot creates additional artefacts to provide visibility in decisions and actions.

Due to the preference of the interviewees to have the prototype provide the users with feedback it will automatically leave visible traces of decisions and actions. Therefore, the best fitting value will be **transparent**.

Reactivity

The reactivity describes the time the bot takes to respond to the user[27]. The possible reactivity values range from synchronous to asynchronous:

1. Synchronous - The bot responds at the same time;
2. Transparent - The bot mixes synchronous and asynchronous response times;
3. Asynchronous - The bot responds after some time has passed.

Since the prototype will be a chatbot, it's response time should be as short as possible. Therefore the prototype in this study should be **synchronous**.

Scheduling

"A bots reasoning scheduling describes the bots strategy for dealing with multiple inputs or outputs that need to be reasoned about" [27]. There are two exclusive states available:

1. Single Tasked - The bot can handle one stimulus at a time;
2. Multiple Tasked - The bot can handle multiple tasks and stimuli at once.

The interviewees state that they do not see the need of the prototype handling two tasks at the same time. They state that a single action should be finished before moving to another task. Therefore, the prototype should be **single tasked**.

Adaptability

Lebeuf et al. describe the adaptability of a bot as its ability to alter or change its behaviour or functionality at runtime [27]. There are two exclusive adaptability states:

1. Non-Adaptive - The bot is not able to change its behaviour at runtime;
2. Adaptive - The bot is able to at least change some behaviour at runtime.

Since the prototype will have two clear functionalities: 1) Reporting bugs and 2) Requesting functionalities it should not change its behaviour at runtime. Therefore the prototype will be **non-adaptive**.

Goals

The goals of the bot are described in this section and are split up in different facets [27].

Complexity

The complexity facet describes how complex the goals of the bot are [27]. The values range from low to high:

1. Low - The bots tasks are simple;
2. High - The bots tasks are complex.

Since the elicitation of requirements from natural language feedback is the main goal of the protype, the tasks it should perform will most-likely be of **high** complexity.

Criticality

The criticality facet describes how critical the goal of the bot is [27]. The values range from low to high:

1. Low - The tasks are of low risk;
2. High - The bots tasks are of high risk.

Since the prototype should always create proto-requirements, and will rely on human control and interaction the criticality of the prototype should be **low**.

Attainability

The attainability facet describes how attainable the goal of the bot is. The boolean values are:

1. Achievable - The bot's goal is achievable and has an explicit end-state;
2. Homoeostasis - The bot's goal is not achievable and has no explicite end-state.

The interviewees state that the prototype should be able to detect when it has gathered enough information to create a new request or report. Therefore, a clear end-state of the prototype can be determined. Due to this fact the goals of the prototype will be **achievable**.

Explicitness

The explicitness facet describes how explicit the goals of the bot are defined [27]. The boolean values are:

1. Explicit - The bots goals is clearly defined, described and with no room for interpretation;

2. Implicit - The bots goals are not clearly defined but instead ambiguous.

The goal of the prototype is to elicit requirements from natural language. Therefore, the goals of the prototype will most likely be **explicit**.

Source

The bots goal source describes where the bots goals originated from [27]. The boolean values are:

1. Internal - The bots goals are derived at runtime or provided in code;

2. External - The bots goals are adopted from external stakeholders.

The goals of the prototype should be derived both from code and external stakeholders. Therefore, **both values** should be true.

Delegation

Lebeuf et al. describe the delegation as "its permission or authority to act on behalf of or to represent others" [27]. The values range from none to complete:

1. None - The bot does not have the authority to act on behalf of others;

2. Partial - The bot has authority to do things on behalf of the user, but does not pretend to be the person its representing;

3. Complete - The bot has the authority to both act on behalf of and pretend to be the user themselves.

The prototype should not act on behalf or pretend to be a user, so the delegation value should be **none**.

Specialization

The bots specialization is the degree to which the bot focuses its efforts in a specific area. The values range from generalist to specialist:

1. Generalist - The bot supports a wide range of tasks;

2. Specialist - The bot focuses on a specific tasks.

As the interviewees have stated that the prototype should focus on a specific set of tasks. Therefore, the prototype should be a **specialist**.

Antrophomorphism

A bots level of anthropomorphism is the degree to which the bot has been given human-like characteristics or traits. [27]. As the interviewees have stated that it should be clear that the prototype is a bot, it should not have any human traits or a personality. Therefore the value for all sub-facets in this category should be **none**.

Life Cycle

Life cycle describes the various phases the bot goes through in it's life [27].

Creation

The creation of the bot is described as the way the bot was brought to life [27]. There are three exclusive states:

1. Human - The bot was created by a human;
2. Bot - The bot was created by another bot;
3. System - The bot was created by another system.

As the prototype should be initialized by a human it will be created by a human. Therefore, the state here should become **human**.

Lifespan

The life span of the bot is described as the time the bot continues to function [27]. There are three exclusive states:

1. Terminating - The bot has a terminating lifespan;
2. Transient - The bot passes in and out of existence;
3. Continuous - The bot never self-terminates.

As stated by the interviewees the prototype should self-terminate at some point, end the conversation and should appear again when needed. Therefore, the state should be **transient**.

Reproduction

The bots reproduction describes the bots ability to spawn other bots [27]. There are two exclusive states:

1. None - The bot is not able to create new bots;
2. Reproductive - The bot is able to create new bots.

The bot in this study should not be able to create new bots. Therefore, the state is **none**.

3.4.7 Interaction Dimension

The interaction dimensions consists of seven facets displayed in Figure 3.13 and describes the bots interaction with different elements in its environment [27].

Access

The access facet described the degree to which the bot has access to its environment, aiming at the restrictions the bot has to its environment and gives us three exclusive options [**?**]:

1. None - The bot is not allowed to access any of its environment;
2. Partial - The bot is allowed to access some of its environment;
3. Complete - The bot is allowed to access all of its environment.

In our case the prototype should have **partial** access as it should be allowed to access most but not all of the environment in the software product, due to security and confidentiality.

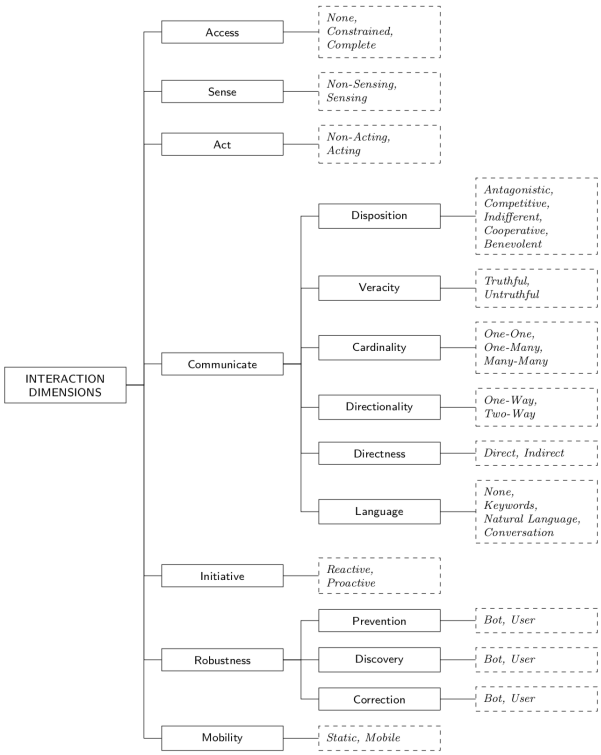


Figure 3.13: Interaction dimensions [27].

Sense

Lebeuf et al. describe sensing as the "degree to which the bot is able to perceive stimuli in its environment." and gives us two options [27]:

1. Non-sensing - The bot does not try to perceive external stimuli;

2. Sensing - The bot tries to perceive a limited set of stimuli in its environment.

The prototype should be **Not sensing** as it will not contain any kind of sensors and thus is not able to sense stimuli in its environment.

Act

The bots acting refers to the ability to act or make changes to its environment and consists of two exclusive states [27]:

1. Non-acting - The bot does not try to act upon or make changes to its environment;

2. Acting - The bot tries to act upon or make changes to its environment.

The prototype will be **Non-acting** as it should not try to act upon or make changes to the environment it is in. It simply elicits requirements from feedback and does not make any changes on its own.

Communicate

Communication is described as the bots ability to meaningfully interact with others in it's environment [27]. Lebeuf et al. state that there are two exclusive states available: 1) non-communicative and 2) communicative [27]. The prototype will be able to interact with others in it's environment it will be **communicative**.

Disposition

Disposition is described as the willingness of the bot to help, perform actions for, or share resources with others in the environment [27].

1. Antagonistic - A bot can be antagonistic if it makes purposefully antagonistic attempts to inconvenience or undermine others [27];

2. Competitive - A bot is competitive if it acts in favour of its own self-interests [27];

3. Indifferent - The bot is indifferent if it is unaware of the needs of others (inadvertently or by choice) [27];

4. Cooperative - The bot is cooperative if it is willing to help others in its environment, potentially sacrificing its own goals [27].

5. Benevolent - The bot is benevolent if it always helps others in its environment, even if it is detrimental to its own goals or best interests [27].

Since the prototype aims at helping others in its environment at all times, we can assume that it is **benevolent**.

Veracity

Lebeuf et al. describe veracity as "the bots deliberate adherence to or divergence from the truth during its communications" [27]. They offer three different values:

1. Untruthful - The bot is untruthful if it tries to deceive the users;

2. Mixed - The bot is mixed if it tries to deceive the users at some times and is truthful at others;

3. Truthful - The bot is truthful if it does not attempt to deceive the users.

The interviewees all state that the both should be honest that it is a bot and should not try to deceive the user in any way. Therefore, the prototype will be **truthful**.

Cardinality

The bots cardinality describes the number of users it can interact with at the same time. Lebeuf et al. describe three different boolean values for this:

1. One-One - The bot is only capable of interacting with one individual at a time;

2. One-Many - The bot is capable of interacting with many users at the same time;

3. Many-Many - The bot is capable of interacting with many users at the same time and they are able to interact with each other as well.

Since the prototype should be capable of interacting with a single, and many users at the same time both the **One-One and One-Many** cardinality values should be true.

Directionality

The bots directionality describes the way the bot communicates with its environment and the following values are available:

1. Indirect - The bot communicates indirectly;
2. Direct - The bot communicates directly with others.

The only way the prototype communicates with end-users through direct-messages. For this reason the value for directionality will most likely be **direct**.

Language Capability

The bots language capability describes it's conversational capabilities and the way it communicates with humans [27]. The following values are given:

1. None - The bot is not able to use human language;
2. Keywords - The bot is only able to communicate using keywords and predetermined phrases;
3. Natural Language - The bot is able to communicate using natural language;
4. Conversation - The bot is able to engage in meaningful two-way dialogues.

The interviewees state that they would like to communicate with the prototype using natural-language, and that it should be able to detect entities and phrases using natural language processing. Therefore, **natural language** is the value that should be chosen.

Initiative

The initiative describes the way in which a bot initiates interaction with its environment and has two values [27]:

1. Reactive - The bot initiates actions in response to a stimuli;
2. Proactive - The bot takes control of the situation rather than responding.

As the prototype proposed should be able to gather feedback in the most unobtrusive way, we want the user to initiate a conversation rather than the prototype starting one. Therefore the value for initiative would be **Reactive**. However, the results from the interviews point out that a combination of both is the preferred way. The prototype should be reactive most of the time and should be proactive when a bug or error occurs taking control of the situation.

Robustness

In this section the robustness of the bot, the ability to detect, handle and correct errors or ambiguity, is described. It is broken down in three different sub-facets: 1) Error prevention, 2) Error discovery, 3) Error correction [27]. The requirements of this part of the prototype are elicited from the interviews with the developers.

Error prevention This value describes the bots ability to reduce or prevent input errors from users [27]. The interviewees state that the main responsibility of this should be at the side of the prototype. However, they state that at certain points the user has a degree of responsibility to provide the prototype with valid input.

Therefore, both the **chatbot and the user** should be responsible for error prevention.

Error discovery Error discovery is described by Lebeuf et al. as the strategy that the bot uses to detect errors in the input [27]. The interviewees agree that the bot should be able to detect and discover errors autonomously. In addition, they state that the prototype should provide the user with feedback about it's interpretation and therefore partly relying on the users ability to discover errors.

Therefore, both the **chatbot and the user** should be responsible for error discovery.

Error correction Lebeuf et al. describes the error correction as the strategies the bot uses to recover from a detected errors [27]. The interviewees state that both the user and the bot should be responsible for correcting errors. The bot should attempt to correct an error by using a knowledge base and the bot should offer the user the possibility to correct the error.

Therefore, both the **chatbot and the user** are responsible for error correction in this study.

Mobility

A bots mobility describes its ability to move around within its environment. A bots degree of mobility can vary for different dimensions. A bot could be mobile in where it interacts (e.g., acts, senses, communicates) and/or where it reasons. For the scope of this taxonomy,

The bots mobility describe the ability to move around within its environment and has two exclusive options:

1. Static - The bot is not able to move within its environment;

2. Mobile - The bot is able to move within its environment, for example a slackbot moving between channels.

As the prototype will reside in a software product and gathers feedback, it will have a static position and should not have the need to move. Therefore, our prototype should be **Static**.

3.4.8 Human-Chatbot conversations

Hill et al. have researched the differences between human-human and human-computer interaction, they have compared 100 random instant message conversations between humans and 100 random conversations with Cleverbot, which is a chatbot [21].

They found that a human writes fewer words per message when sending messages to a chatbot than when they were conversating with another human-being. However, they did send more than twice as many messages to the chatbots as they did to another person [21]. They also confirmed there hypothesis that the anonymous nature of human-computer conversations would lead to more profanity in the messages. There research showed that human-computer interactions contained a lot more profanity than the human-human conversations and were also very sexually explicit, contained more sear words and negative emotion words [21]. While cleverbot is a social chatbot and not a chatbot that is used in professional conversations this has to be kept in mind when developing the prototype. They concluded that while chatbots are limited to communicate with humans socially and share common experiences, many people are still willing to communicate with and have extensive conversations with chatbots. However, they do not seem to be able to have a fully intelligent human, socially centered, conversation just yet [21].

3.5 Prototype Requirements

The requirements in Section 2.1 merely provided us with the initial requirements for the prototype, the information gathered from the semi-structured interviews and described in Section 3.4 can be turned into requirements to complete the list of requirements the prototype has to satisfy.
In Appendix B the complete overview of the bot's taxonomy is displayed. This taxonomy was derived from the interviews with the consultants at ChipSoft in combination with the literature review. The taxonomy can then be used as a guideline for the additional requirements of the prototype. We will highlight and explain the most important requirements that can be derived from this taxonomy split per dimension.

3.5.1 Environment dimensions

The following requirement result from the environment dimensions:

- **R8 - Integrated in application** The working service should ultimately be integrated within a software product, in this case HiX.

As all the facets address characteristics about the environment of the prototype, the most important requirement that can be derived from this dimension is that it should be integrated in a software application, in this case HiX. At the same time this requirement, due to the way the application is built, the prototype to-be is **bound**, **closed**, **dynamic**, **uncertain** and the permanence of the changes are **sequential**. In addition, due to the countable number of users in HiX and the fact that there are various user-groups it also covers the fact that the prototype should be **countable** and **heterogeneous**. However, due to security and time constraints we will not be able to realize this prototype. The reason for this is that the prototype will never pass the security requirements in the time we have for this research at the case study. Therefore, we have chosen to not satisfy this requirement but rather describe the possibilities of implementing the prototype in an application.

3.5.2 Intrinsic dimensions

The following requirement result from the intrinsic dimensions:

- **R9 - Conversation storing** The prototype should save conversations for the short- and long-term;
- **R10 - Provide feedback** The prototype should provide the user with feedback to create transparency regarding the decisions made;
- **R11 - Sufficient information detection** The prototype should be able to detect when it has gathered enough information to create the proto-requirement and transition to an end-state;
- **R12 - Human initialization** The prototype should only react to a human initializing the prototype;
- **R13 - Transient lifespan** The prototype should be able to self-terminate itself when the end-state is reached and should appear again when a user initializes it.

The facets in this dimension point out the internal properties of the prototype itself. The most important requirements that can be derived from this dimensions are the ones that describe the behavior and information/knowledge handling of the prototype. In addition, it is important to have a clear view on the lifespan and initialization of the prototype itself. The **source**, **mechanisms**, **agency**, **reactivity**, and **adaptability** facets are already handled in the initial requirements as stated in Section 3.5.

3.5.3 Interaction dimensions

The following requirement result from the interaction dimensions:

- **R14 - Provide user with clarity** The prototype should state to the user that it is in fact a bot and not deceive a user that it is human;
- **R15 - Support multiple sessions** The prototype should be able to communicate with multiple users at the same time, this goes for multiple independent conversations simultaneously and the prototype should not support multi-party conversations;
- **R16 - Natural-language** The prototype should be able to detect and converse in natural-language;
- **R17 - Error-prevention** The prototype should be able to assist the user in error discovery, prevention and correction.

The facets in this dimension point out the way the prototype should communicate with users. The most important requirements that can be derived from this dimensions are the ones that describe the way the prototype should act, communicate and handle input errors with users.

Chapter 4

Research Method

In this chapter, we discuss the research method including the research questions, case description and experiments. The research method consists of a combination of desk research, prototype design & development and a case study. The case study is executed at the company ChipSoft B.V. based in Amsterdam. ChipSoft is a leading software company in the healthcare industry in the Netherlands. They have developed a software product named HiX (Health Information X-change) which is an innovative healthcare product, that provides a complete integrated solution to hospitals, pharmacies, general practices and other healthcare industries. In this thesis project, we will tailor the prototype to HiX as explained in Section 2. The research question that will be used in this thesis project goes:

"How can chatbots integrated within a software product prove beneficial in eliciting requirements from user feedback?"

This leads to the following sub-questions:

1. **RQ 1 -** How can users be motivated to provide feedback through a chatbot integrated into a software product?
2. **RQ 2 -** How can chatbots help guiding users in providing informative feedback about the software product?
3. **RQ 3 -** How can proto-requirements be automatically elicited from natural-language user feedback?

4.1 Research questions

The goal of this thesis project is to uncover how the feedback of large heterogeneous groups of users can be continuously included in the requirements engineering process, by using chatbots integrated in the software product. This can help companies involve the users and provide them with a continuous stream of valuable feedback to be used in the requirements process. The company can use the feedback to elicit requirements and tailor the software product to the users needs more accurately. Additionally, the conversations between the chatbot and the users can be analyzed to improve the understanding of the users' needs and wishes, resulting in a more successful software product and greater user satisfaction.

The first research questions aims at uncovering how users can be motivated to interact with a chatbot that can be integrated in a software product, this revolves mostly around user involvement. The second research question provides an answer about how the interaction between users and a chatbot should be, to guide them in providing valuable feedback. Finally, the third research question focuses on extracting information and transforming the feedback to proto-requirements,

proto-requirements are prototype requirements that can be used as the base for defining actual requirements. Pohl describes that to reach a complete specification of requirements one has to reach a certain maturity in: 1) Specification, 2) Agreement, 3) Representation. Specification is focused on the way requirements are specified using a formal language and representing requirements in a certain formal language [35]. This can be achieved by creating a prototype that offers the possibility to elicit requirements and specify them in a formal language from natural language. However, the prototype can not offer the possibility to reach a higher maturity level on the agreement axis as displayed in Figure 4.1, without human interference. In addition, the representation can not be adjusted by the prototype and always needs human interaction [35]. Therefore, the only possible output of the prototype will be proto-requirements as human interaction is needed to reach complete specification and maturity of these requirements.

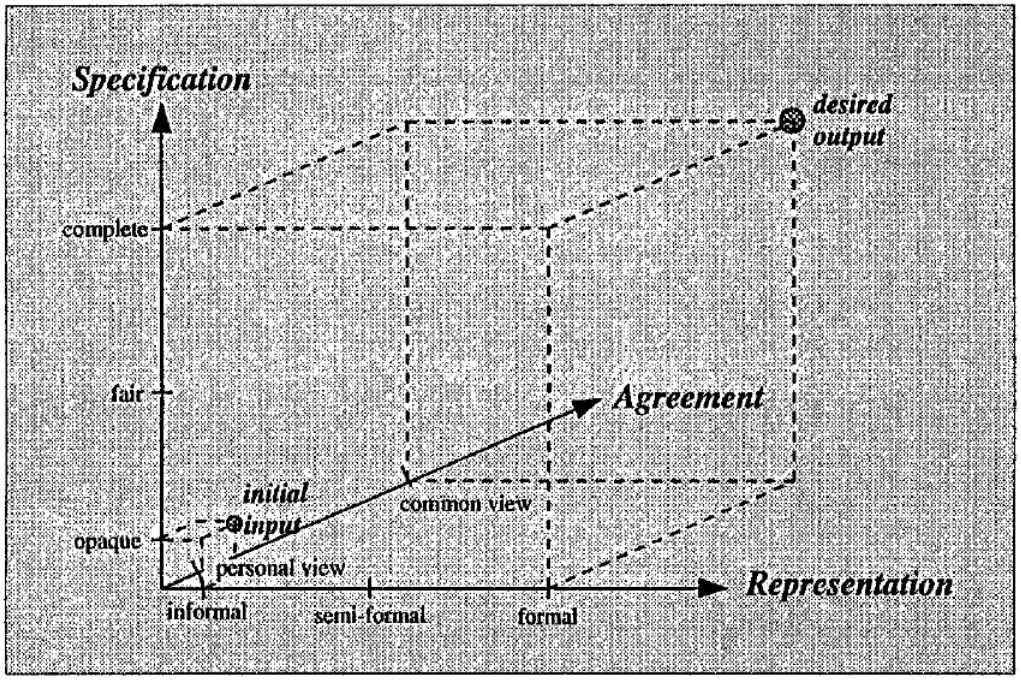


Figure 4.1: The Three Dimensions of Requirements Engineering [35].

4.2 Method

To answer the research questions presented in Section 4.1 we have chosen to use qualitative research. The reason we have chosen for qualitative research is because of the exploratory nature of the research at hand. Because of the fact that a prototype of a chatbot will be developed, the design science methodology will be used throughout this thesis project.

RQ 1 will be answered through the literature review in which we gather information about human-chatbot interaction and how to motivate users to provide the company with a continuous stream of valuable feedback about the software product. In addition, the interviews with the stakeholders provides us with more insights in how they would like to communicate with a chatbot. RQ 2 shall be answered by interviewing the stakeholders and researching how chatbots can help users in providing valuable feedback to create a prototype fitting to the case at hand. Additionally, conversation flows will be created and discussed with domain-experts to evaluate them. Finally,

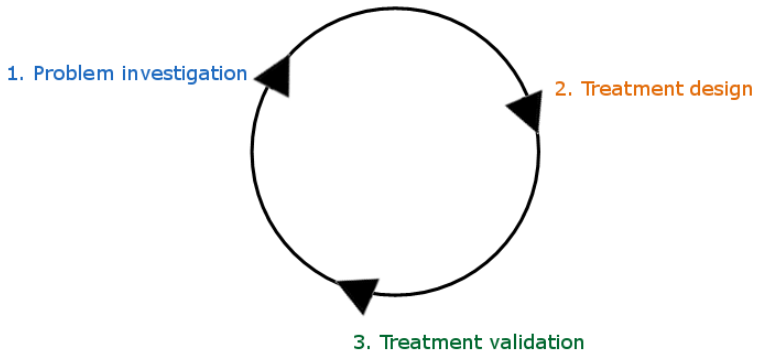


Figure 4.2: Design science cycle [45].

RQ 3 can be answered by designing & developing a prototype that automatically elicits proto-requirements from feedback provided by users in natural-language, and evaluating the quality of these proto-requirements with the stakeholders.

4.2.1 Problem investigation

The first phase in the process is to gather background information and get a clear view of the problem by executing a literature review. In addition, the stakeholders, in this case the consultants and developers at ChipSoft, will be interviewed using a semi-structured interview. By combining the results of the literature review with the results of the interviews, a clear view of the problem will be acquired specifically for this case as described in Chapter 2.

4.2.2 Treatment design

By executing interviews with the stakeholders, the requirements specific for the case will be defined. In our case, the stakeholders are the consultants and developers, who will use and validate the prototype. In addition, the prototype will be designed and developed in this phase. The design and development of the prototype will be executed in short iterations. In each iteration a part of the prototype will be designed and developed followed by a validation with the stakeholders. We have chosen this approach to be as flexible as possible and tailor the prototype to the stakeholders wishes as accurate as possible. For the development of the bot ChipSoft has stated that they prefer to use the Microsoft Bot Framework, for this reason we will not look into other frameworks for the development of the chatbot prototype. This is a pragmatic consideration that yields a constraint for the design of the prototype, this constraint leads to less freedom in defining the prototype and designing the type of conversations. This is typical for industry-hosted research where the environment has to be taken into consideration when developing and defining the treatment.

4.2.3 Treatment validation

The prototype will be evaluated and validated with the consultants and the developers using interviews and a comparison of the situation with and without the prototype implemented.

4.2.4 Chatbot purpose

To clarify the purpose of the chatbot in the case study we have held semi-structured interviews with four different consultants and three different developers at ChipSoft. These interviews focused on gathering the opinions and the requirements for the chatbot. In addition, focused on clarifying the purpose of the chatbot. The interviewees included three I&S (Implementation and

Support) consultants, the I&S Teamlead of the Multimedia consultancy team, two Software developers and the Teamlead of the Multimedia development team.

The interviewees state that bug reports most of the time originate from three sources: 1) Feedback during projects, 2) Support calls, 3) Meetings between consultants and developers. The bugs are then described in a software request.

Figure 4.3: Structured vs. unstructured requests

However, the interviewees state that these processes differ a bit per module at ChipSoft. Some of the modules focus more thoroughly on the description of the problem where the Multimedia module relies on the flexibility and speed of the communication between the developers and consultants. However, this results in less organized registration of the requests and reports. The consultants state that there is no central document or place for registering and looking up the feature requests and bug reports which results in loss of control. In contrast, the developers state that they would like to have more structure in the requests and reports. This is supported by researching the current available software requests and pinpointing which were created conform the software requests structure as stated in Section 5.1 displayed in Figure 4.3. This research only contained the available software requests that contained a bug up until 22 October 2019. As displayed, there is still a large percentage (48.1%) of software requests that are not conform the software request structure. In addition, the consultants suggest creating a digital platform for storing all the requests and reports for a better overview. In addition, the description of new features require a lot of technical insights from the consultants, which in turns leads to discussion between developers and consultants and increased work.

The consultants state that at some point a feature request or bug report might be rejected. However, the reason why these get rejected is too varied to make a clear statement about the most common reason behind these rejections. The reasons range from duplicate entries, time constraints and wrong reasoning of the solution suggested. The developers however state, that the requests get rejected mostly because of the request not being clear enough. The consultants do state that further clarification of the requests would not lower the rejection rate. In contrast, the developers all state that further clarification and a better description of the request would result in a lower

rejection rate.

The interviews unveiled that the chatbot should provide a way to create a better overview of all the requests and reports and provide the consultants with a set of proto-requirements for the consultants to start with. It has to focus on providing the consultants with more insights in the users wishes and a clear description of the problem or missing feature and the expected end-results. When reporting a bug the chatbot should use the information available from the software application without having to question the users. In this, the chatbot should also be able to categorize the requests/reports per functionality making it easier to prioritize them and increasing the overview. The developers would like to see more structure and as much information as possible in the requests, and check if there isn't a duplicate request at that point.

The consultants see the chatbot as an addition to the current support site that is used to avoid duplicate data as much as possible. It should be able to replace a part of the support and handle the elicitation of basic information and grant ChipSoft with more insights in the users wishes. The developers would use the chatbot as a way to elicit more requirements and providing them with more information.

Chapter 5

Conversation flows

This chapter describes the conversation flow that will be used in the prototype. The conversation flows have been created in iterations and serve as the starting point for designing the prototype. These have been designed using an experts' opinion at ChipSoft by executing a semi-structured interview after each iteration. The chapter will go through every iteration of the conversation flow and explain how this iteration was executed split up in different sections.

5.1 Software Request Structure

We will explain how the structure of a software request looks at ChipSoft. A software request (SIF) protocol/template is used within ChipSoft to describe the structure a software request should have to be accepted by ChipSoft. At the very least a SIF should contain a **problem statement and desired end-result** according to the interviewees, and the software request protocol. However, in the current situation the software request protocol is not used in most situations. The interviewees confirm this by stating that they miss structure in the current software requests. The interviewees state that in case of a bug a **reconstruction scenario** could be valuable for uncovering the cause of the bug and further specifying the SIF. Finally, the interviewees state that information about the **affected users, location, environment and priority** could prove useful when reporting bugs. In short, a SIF regarding a bug should contain the following elements:

- **Problem statement** - The problem statement should describe the bug experienced as detailed as possible and is the starting point of a SIF;
- **Desired end-result** - The desired end-result should describe how the user expects the application to work after fixing the bug on a functional level;
- **Reconstruction scenario** - The reconstruction scenario should describe the process leading up to the bug step-by-step. This provides the developers and consultants with a detailed description of the problem and an indication on where it occurs;
- **Affected users** - The affected users describe the user group that is affected by the bug mostly. This information can also be extracted from within the application itself;
- **Bug location** - The bug location describes where the bug occurs in the application. Additional information can be gathered using information residing within the application;

We will go over each of the elements and explain what information should be attained for each of these elements.

5.1.1 Problem statement

The problem statement should define the problem as concrete and accurate as possible. More precise, as Max Kush states it should identify the gap between the current situation and the desired end-results [25]. To identify this gap, a common approach is to use the 5W's as used in journalism [19]:

- Who - Who does the problem affect?;
- What - What is the issue?;
- Why - Why is it important to fix the problem?;
- Where - Where is the issue occurring?;
- When - When does the issue occur? [28].

By answering every question stated in the 5W's method, we can clearly identify the problem at hand. In addition this helps in filling the **affected users and bug location** elements. It seems clear that when reporting a bug the starting point should revolve around describing the bug experienced. This why we have chosen to start the conversation flow with the "what" question as can be seen in Appendix D.

5.1.2 Affected users

The affected users should describe the "who" of the 5W's method, and give a clear description of the users or user groups that are experiencing the problem at hand. In this case there is a fixed list of user groups dependable on the implementation of HiX at the customer. Additionally, this element might already be extracted from the problem statement, if this is not the case, the follow-up question to discover the "who" should be asked.

5.1.3 Bug location

The bug location describes the location where the bug occurs, more specific it should uncover the functionality and screen that is open at the time of the bug occurrence. In some of the cases this element might be extracted from the answer given to the problem statement question, if that is not the case, the follow-up question to discover the "where" should be asked. Again, additional information about the bugs location can be extracted from within HiX without having to burden the user with more questions.

5.1.4 Reconstruction scenario

The reconstruction scenario is a step-by-step walk through up until the moment the bug occurred. This element should be used to give the developers some guidance about the root cause of the bug. In addition, information from within HiX, such as the logging, might be used to provide valuable information about the bug occurrence.

5.2 Initial conversation flow

The initial conversation flow as displayed in Apendix D have been created after conducting the semi-structured interviews with the consultants at ChipSoft. At first, we wanted to make a clear overview of the structure of a conversation between a chatbot and a human. As stated in Chapter 3.5 the bot should make clear that the user is communicating with a bot instead of a human. For this reason the "welcome message" contains a sentence that clarifies that the user is talking to a bot. In addition, the bot only reacts when a human initializes the chatbot, satisfying R11 and R14 as stated in Chapter 3.5.

The next logical step was to classify if the bug experienced is production blocking for the user. If this is the case the user should be redirected to a human employee if they want to. The reason why we have not implemented a feature requesting scenario in the conversation flow is that, the interviewees stated that reporting a bug could prove more valuable than requesting a new feature as the chatbot could use valuable information from within HiX to enrich the bug report. In this chapter we will go through each of the questions step-by-step to clarify why we added these question to the conversation and how they relate to the requirements as defined in Section 2.1 and Section 3.5. The full conversation flow can be seen in Appendix D.

5.2.1 Production blocking question

As the interviewees have stated there should be a way to rank the urgency of a software request as described in R3 in Section 2.1. While some bugs might be more urgent than others, there are also bugs that lead to production blocking situations for the user. This means that the user is not able to advance in a certain work process while experiencing the bug. In that case, as the interviewees have stated, there should be a fail safe mechanism that can connect the user directly to a consultant. If at some point there is no consultant available, the user is asked if they want to continue the report through the bot or cancel the report. If the user wants to continue the conversation, the bug report will be stored as production blocking and will be given the highest priority available. Since this relies on the user reporting the bug being entirely honest instead of detecting if the bug is production blocking or not, this is not entirely safe. In addition, for a user to understand what is production blocking and what is not, requires some experience and proper knowledge of the system. However, for this prototype and the test setup that uses the consultants at ChipSoft, which have a solid understanding of the product and know what bugs are production blocking, this mechanism is sufficient. We expect that for this mechanism to work at users a further classification and check, needs to be added to verify if the bug is production blocking. This part of the conversation will not be added to the actual prototype due to time constraints and the employees in case are the actual subjects. The full conversation flow can be seen in Appendix D.

5.2.2 Uncovering the problem

The next step in the conversation is to uncover the “what” of the problem at hand, which relates to the problem statement element of a software request as explained in Section 5.1. The bot will question the user to describe the problem they are experiencing using the 5W’s as explained in Section 5.1. The first step in extracting all the W’s is to gather information about what problem the user is experiencing. In a lot of situations the user might already reveal other elements such as the “who” or “where/when” of the bug at hand. In these situations those questions could be skipped, this keeps the conversation flow as short as possible and reduces the redundancy of questions. If these elements can not be extracted from the answer to the “what” question they will be asked in the sequence displayed in Appendix 5.2 depending on what element is missing. This whole question mechanism can satisfy R1 as described in Section 2.1.

At this point the “why” question is still included, but we have noticed that the “why” of a bug is most of the time that something is not working properly. The same goes for the “when” and “where” questions, as we think that they may point to the same thing which likely is the location where the bug occurs. There is a very thin line between these elements meaning that they might be combined in later iterations.

Finally, the bot will summarize the request and returns the summarized request to the user. At this point the user can review the created request and make changes to it, which in combination with the extraction of the elements satisfies R5 and partly satisfies R6 and R10 as described in Section 2.1 and Chapter 3.5. The full conversation flow can be seen in Appendix D.

5.2.3 Question expected end-result

When the problem statement has been extracted the next step is to clarify what the user expects/desires of the end-result. This part of the conversation just records what the user answers to this question. In addition, it questions if the desired end-result changes the current way of working for the users. If so, it questions what changes in the current way of working. This part of the conversation can satisfy R2 as described in Section 2.1. The full conversation flow can be seen in Appendix D.

5.2.4 Reconstructing the scenario

The reconstruction scenario should be questioned to provide the developers with a way to reconstruct the scenario at the given customer and fill the element in the software request structure. This part of the conversation can satisfy R4 and partly R7 as described in Section 2.1. The idea is to let the user describe the process that led up to the bug step-by-step. We expect a video or a screen capture might be more valuable, but this is not within the scope of this research and the case study at hand. Therefore, we have chosen to go with a question/answer mechanism in which the user describes every step and the bot finally summarizes and creates a list of these steps. Furthermore, the bot will eventually gather information about the user, the environment, current open screens, call stack, etc. to add more value to the reconstruction scenario. Finally, the bot summarizes the extracted reconstruction scenario and reports it back to the user validating the report. The user can then choose to add, delete or edit the reconstruction scenario. The full conversation flow can be seen in Appendix D.

5.2.5 Mapping the urgency

During the entire conversation an urgency score can be stored depending on the answers given by the user. This score is calculated using the information available inside the application and the answers provided by the users. This information includes, the location of the bug, the user experiencing the bug, impact of the functionality failing and other factors. The urgency score ranges from 1 to 5 where 1 is the highest urgency score and 5 is the lowest. At this point, the user will be asked if the mapped urgency is correct, if the user agrees with the mapped urgency score this score will be taken. If the user disagrees with the urgency score, the user will be given the choice to raise the urgency score by 1 or lower the urgency score by 1. This gives the user some influence in the urgency score while preventing them from always choosing the highest possible rating. This process partly satisfies R3 as described in Section 2.1, as it can classify the bug in terms of urgency. The full conversation flow can be seen in Appendix D.

5.2.6 Summarize request

After mapping the urgency the entire request will be summarized and displayed to the user to validate the report. The user can then choose to edit the report if they disagree with the summarized report. This summary satisfies R6 and R9, as described in Section 2.1 and 3.5, as it provides users with feedback through every step and finalizes the feedback with a full summary of the extracted report. If the user agrees with the summary the report will be stored. The full conversation flow can be seen in Appendix D.

5.2.7 Rate chatbot

The final step, is asking the user to rate the chatbot in terms if it was useful or not. This step of the conversation is for research purposes and helps in understanding the weaknesses and strengths of the prototype. The user will be given the choice to rate the chatbot or not and give some feedback about the bot. The rating will take place through a 1-5 star rating where after the user will be asked to give an optional short description. This function will not be implemented in the current case-study as we will evaluate the effectiveness and usefulness of the bot using semi-structured

interviews. However, if the bot is to be implemented at end-users this feature might prove useful. The full conversation flow can be seen in Appendix D.

5.3 Conversation flow adjustments

After having consulted the chatbot expert at ChipSoft, and through conversation and feedback with the first supervisor of this thesis project, some adjustments have been made to the initial conversation flow as described in Section 5.2. This chapter will go through these adjustments and explains why these changes have been made.

5.3.1 Question problem adjustments

After having discussed the initial conversation flow with the first supervisor of this thesis project and the chatbot expert at ChipSoft, we came to the conclusion that the order in which the elements were filled in the initial flow was not optimal. In addition, there seemed to be a lot of redundancy in questions and questions that did not add as much value as thought at first.

The first change was to reduce the amount of questions in the "what" part. The reason behind this change was, that the questions about what would happen if the bug is fixed and what would happen if the bug is not fixed do not add any value to the element. In addition, we are running the risk of the dialogue becoming very boring and way too extensive. Furthermore, we are now focusing more on extracting the "who" and "where" element from the answer provided to the "what" question further automating this process and therefore simplifying the conversation for the user.

The next change was to further split the "what" and "why" elements and removing the "why should this problem be solved" question. The reason behind this is that the why should the problem be solved question is theoretically the same question as the how does this problem limit you in your work question. By removing the redundant question the conversation gets simplified and less of a drag to the user.

We then found out that the "where" and "when" dimension are quite similar when reporting a bug. The reason behind this is that they both lead to a location in the application where the bug occurs, for this reason we have chosen to combine these dimensions in a single question. Again, reducing the redundancy and burden of the conversation for the user.

Because we now focus more on automatically extracting and detecting the "who" and "where" dimension from the answer provided in the "what" question, we decided to validate the extracted elements. If the bot extracts a "who" or "where" element from the "what" question the bot will first validate if this element was correctly detected and extracted. If this is the case the bot will only question the missing elements, therefore again reducing the burden of the conversation and further automating the process. This also increases the degree of control we have on the conversation as we can further define where the conversation is going. In addition, this change helps further satisfying R11 and R19 as described in Section 3.5.

Chapter 6

Prototype design

This chapter describes the design of the prototype that was implemented during the case study at ChipSoft as described in Chapter 2. This chapter first explains the enabling technology used in the prototype and is then split up in different versions of the prototype where each version represents an iteration in the process of the prototype design. In addition, the reasoning behind the choices made will be explained.

6.1 Microsoft LUIS

For the prototype we have decided to use the LUIS (Language Understanding Intelligence Service) by Microsoft. This service is a machine learning-based service that supports natural language understanding an processing (NLP & NLU). The service identifies different user goals in so-called "intents" and extracts valuable information from sentences in the form of "entities". The service supports 13 different languages is available worldwide and implements active learning to continuously improve the quality of it's models [1]. In addition, it seamlessly integrates with the Azure Bot Service and C# and thus, with the code base of HiX. Another reason for using LUIS instead of other NLP/NLU frameworks is that the chatbot expert at ChipSoft stated that ChipSoft already uses LUIS for their other chatbot, this means that there is more knowledge available within the company. In contrast, we have compared the LUIS solution to pattern-matching, which will be discussed in Section 6.1.1. By implementing LUIS into the prototype we can support natural-language conversations and therefore satisfy requirement R18 as defined in 3.5.

6.1.1 LUIS vs. Rule-based pattern-matching

The SpaCy framework offers the possibility to use their rule-based pattern-matching engine, this engine is comparable to regular expression but is based on token-based matching. This token-based matching is done after the PoS (Part of Speech) tagging and lets you find certain words and phrases that you are looking for. They state that rule-based matching system are a good choice when there is a finite number of examples or there are clear, structured patterns such as IP addresses, country names or URL's. While this seems like a good approach at first, the degrees of freedom that a user has while conversing with a bot in natural-language increases the difficulty of identifying clear patterns. Another downside of this approach is that detecting these patterns will be very time consuming and are set in stone afterwards resulting in less flexibility and scalability.

We have researched and identified a number of patterns using the rule-based pattern-matching approach, and measured the accuracy of these patterns given different existing software requests. This approach relies heavily on dependency types identified by PoS tagging, this means that a pattern is identified by looking at the structure and dependencies in the given sentences. To test this approach we have identified a number of patterns by looking at the way existing software re-

quests are built up, and looking at the dependencies between the different elements. These results are displayed in Appendix C. However, we do realize that these existing software requests are not representative for natural-language conversations which most-likely will be split up in multiple smaller answers to the questions as described in Chapter 5. As we can see, the identified patterns that are most accurate are quite general such as "NOUNS", "VERBS" and "ADP". This will result in identifying a lot of entities and words in a sentence, so to filter out the noise an extra filtering step needs to take place. This extra filtering may be, comparing each of the found words to a list of entities available within HiX. This list can consist of entities and synonyms for these entities with different weights, if a certain threshold score is reached the bot will assume that the user means that entity and will extract that entity.

To be able to compare the pattern-matching approach to the LUIS approach more accurately we went through some real-life example answers gathered through a first setup of the chatbot. We have asked a number of developers to exclusively answer the "what" question through the chatbot and avoid using too much technical terms and act if they are end-users. This way we can simulate a number of responses without polluting the experiment population. The reason why we only look at the "what" question is that the answers to this question may contain information and entities about other elements.

To test the LUIS approach, we have created a LUIS application which is able to detect "who", "where", and document type entities using the entity tagging functionality available in LUIS. The "who" and "where" entities are tagged using the machine learning functionality available in LUIS and the document type was implemented as a set list of document types as available in HiX, this list consisted of set document types and their synonyms. To train this model we have added 10 example utterances to the "none" intent and 20 example utterances to the "what" intent. In these example utterances we have manually tagged the entities and trained the application. We have then created a simple chatbot in which the user is asked to answer the "what" question, which then creates a LUIS request and saves the result of the detected entities. Two developers at ChipSoft have been asked to answer the "what" question after each reading 10 randomly selected existing software requests. We have then analyzed the answers and compared them to the pattern-matching approach as will be analyzed further in this chapter.

The goal of this small experiment is to prove that the LUIS approach is more applicable to the case at hand. We will look at which approach is better in detecting the "who", "where" and document type in an answer to the "what" question. Which is better is measured in the amount of false positives, true positives, and false negatives. The SpaCy approach will only take the top 3 of the "where", "when" and "who" patterns with the highest accuracy as identified in Appendix C. However, we will exclude the "NOUN" patterns without a dependency tag as this just identifies all nouns in the sentence and can not be identified as a real pattern.

Example 1

"In the overview screen the function of removing a patient from an object is being called but this has not yet been implemented. Also, too many zismuts are being created when importing an image"

In this example the following entities can be identified:

1. **Patient -** Can be identified as the "who" in the answer as the problem relates to the patient;
2. **Overview screen -** Can be identified as the "where" in the answer as the problem resides in this screen;
3. **Importing -** Can be identified as the "when" in the answer as the problem occurs when importing an image;

4. **Removing -** Can be identified as the "when" in the answer as the problem occurs when removing a patient;
5. **Image -** Can be identified as the document type as the problem relates to an image.

Pattern-matching result We will go through each of the elements and explain what they have tagged and why this is correct or incorrect.

1. **Who -** the blue color represents the top 3 who pattern tagging;
2. **Where -** the red color represents the top 3 where pattern tagging;
3. **When -** the green color represents the top 3 when pattern tagging;

"In the overview screen the function of removing a patient from an object is being called but this has not yet been implemented. Also, too many zismuts are being created when importing an image"

As we can see a lot of the sentence is tagged using the top 3 patterns per element. This also means there is a lot of noise that needs to be filtered out. We can notice that the who patterns also tag irrelevant words, which needs to be filtered out as well. The when patterns seem to miss any of the words referring to the "when" in the answer. We can conclude that the patterns do the job but lack in precision and require a lot of post-processing of the tags. In addition, the pattern-matching needs at least 3 iterations through the answer resulting in slower processing of the answer. Finally, the document type is recognized as a who pattern this is quite normal as there was no specific pattern for the document types.

LUIS result We will go through each of the elements and explain what they have tagged and why this is correct or incorrect.

1. **Who -** the blue color represents the who entity tagging;
2. **Where -** the red color represents the where entity tagging;
3. **When -** the green color represents the when entity tagging;
4. **Document type -** the yellow color represents the document type entity tagging;

"In the overview screen the function of removing a patient from an object is being called but this has not yet been implemented. Also, too many zismuts are being created when importing an image"

As we can see, the LUIS approach is way more accurate and produces less noise in this example. In addition, it correctly detects the document type. The down part of the LUIS approach is that it missed the "removing" part of the "when" element but this may be caused due to a lack of training.

Example 2

"I can not view ecgs in the ecg viewer"

In this example the following entities can be identified:

1. **I -** Can be identified as the "who" in the answer as the problem relates to the current user;
2. **Ecg viewer -** Can be identified as the "where" in the answer as the problem resides in this screen;
3. **View -** Can be identified as the "when" in the answer as the problem occurs when removing a patient;

4. **Ecgs -** Can be identified as the document type as the problem relates to an image.

Pattern-matching result We will go through each of the elements and explain what they have tagged and why this is correct or incorrect.

1. **Who -** the blue color represents the top 3 who pattern tagging;
2. **Where -** the red color represents the top 3 where pattern tagging;
3. **When -** the green color represents the top 3 when pattern tagging;

"I can not view ecgs in the ecg viewer"

As we can see the patterns fail to detect the "who" which is I in this example. In addition, the "when" patterns yield some interesting results as it does not tag anything useful. Again, the "who" patterns also recognize the document type, but this might not be that strange regarding the fact that these patterns have not been split up. Finally, the patterns fail to detect the "when" element in this particular example.

Pattern-matching result We will go through each of the elements and explain what they have tagged and why this is correct or incorrect.

1. **Who -** the blue color represents the who entity tagging;
2. **Where -** the red color represents the where entity tagging;
3. **When -** the green color represents the when entity tagging;
4. **Document type -** the yellow color represents the document type entity tagging;

"I can not view ecgs in the ecg viewer"

The LUIS solution can successfully identify the "who" in this example and successfully splits the document type and the "who". In addition, there is a bit less noise regarding the "who" element tagging.

We have noticed that both of the approaches do the job, but the LUIS approach seems to generate a bit less noise and tag more precisely. In addition, the patterns are set in stone and are not that flexible in contrast with the LUIS approach which keeps on learning its entire lifetime. The LUIS approach also seems to perform a bit better as the pattern-matching requires multiple iterations, post processing and filtering to do the job as efficiently as LUIS does.

6.1.2 LUIS Development cycles

Microsoft has stated some best-practices and created a so-called development lifecycle. This lifecycle explains the process of efficiently creating a LUIS application. The concept of this lifecycle is that each iteration over the full lifecycle creates a new working version of the LUIS application [38] as displayed in Figure 6.1.

The first step in the design process is to build a LUIS app schema, this step is the process of creating/editing the intentions and entities in the application. This step should be followed by defining a couple of training examples or so-called training utterances and features to the LUIS application. The next step is to virtually train the application with the provided example utterances and publish the version of the application. This results in an HTTPS endpoint which can receive utterances through an HTTP request and responds with the extracted intentions through a JSON object as visualized in Figure 6.2.

LUIS mainly consists of three concepts [38]:

1. **Intents -** The intent represents and detects the action the user wants to take;

Figure 6.1: LUIS development cycle [38].

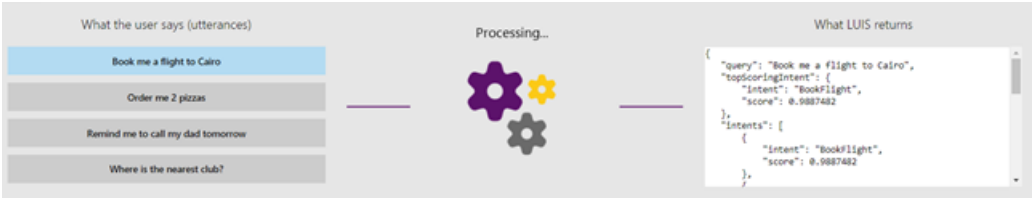


Figure 6.2: LUIS request and response [38].

2. **Utterances -** An utterance is a text input from the user that the application should be able to understand;

3. **Entities -** An entity represents some detailed information that is relevant for a given utterance.

6.2 Microsoft Bot Framework

The Microsoft Bot framework is a framework that focuses on creating intelligent bots, in particular bots that focus on communicating with users through conversations. These bots can be hosted in a so-called Microsoft Azure Bot Service directly in the cloud and seamlessly integrated with Microsoft LUIS as explained in Section 6.1. The bots can communicate with the user through various so-called "channels", a channel can be Facebook, Skype, Slack or the bot can be integrated directly into an application such as HiX by ChipSoft. Every interaction between the user and the bot generates a so-called "activity" which in fact is a HTTP request to the Azure Bot Service which the bot processes and responds to as displayed in Figure 6.3 and 6.4 [42]. Each activity that the bot receives gets routed to the "turn handler" which then in turn links the individual activity to whichever assigned "activity handler" [42]. In addition, one could add "middleware" which are executed in order, giving each the chance to interact with the activity [42]. We will not delve deeper into the technical working of the bot framework but rather focus on the functional structure which we can use to describe the prototype design and choices made during this research. The entire bot framework is asynchronous meaning it can support multiple sessions of independent conversations simultaneously. This means that by using the Microsoft Bot Framework we can directly satisfy requirement R17 and R9 as defined in Section 3.5.

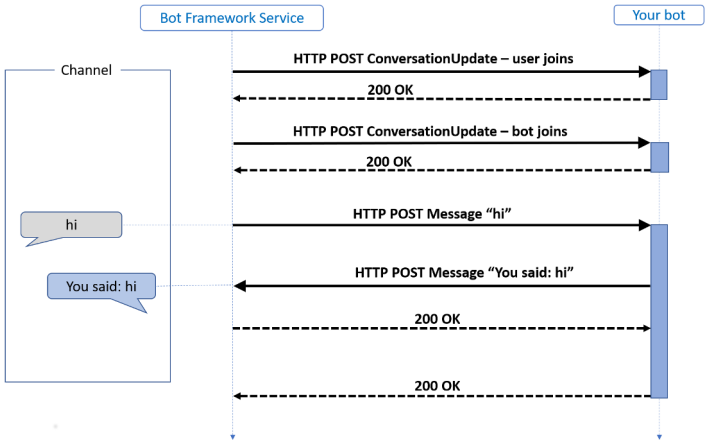


Figure 6.3: Bot builder activity [42].

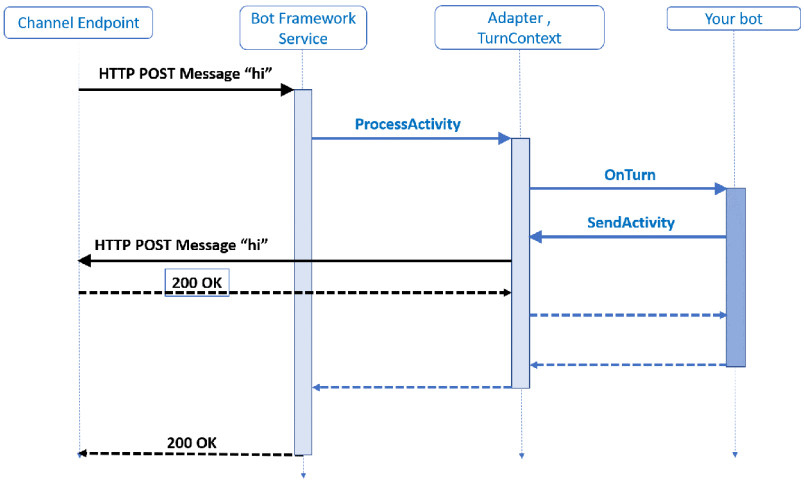


Figure 6.4: Bot builder processing stack [42].

6.2.1 Bot structure

This section will describe the structure and elements a bot, created with the Microsoft Bot Framework, contains. A bot should contain the following elements [42]:

1. **Bot logic -** The bot logic acts as the heart of the bot and processes incoming activities from one or more channels and generates outgoing activities;
2. **Bot controller -** The bot controller follows the standard MVC structure and helps in defining and determining the routing of messages and HTTP POST requests;
3. **Bot resource file -** The bot resource file contains all the resources needed to setup the environment of the bot and connect to the Azure Bot Service;
4. **Dialogs -** A bot can contain several dialogs that define how the bot interacts with a user and how the answers are processed, this also defines the conversation flows of the bot;
5. **Data classes -** Data classes contain information you want to store in an object that can be easily passed between different dialogs.

We will be defining the prototype by describing and explaining the dialogs, bot logics and data classes for each of the development iterations during the thesis project. In addition, we will describe the training process and design of the LUIS application as explained in Section 6.1. This way we will focus more thoroughly on the functional design of the prototype rather than the technical design.

6.3 First prototype cycle

The first prototype cycle focused on correctly identifying, extracting and labeling entities and intents regarding the "what" question. We will first go through the bot design followed by the design of the LUIS application the training of the model and test results.

6.3.1 Bot Design

The initial design of the bot itself has been created, this includes the following **dialogues** as displayed in Figure 6.5:

1. **Welcome dialog -** This dialog states the welcome message as defined in Appendix D and helps;
2. **Main dialog -** This dialog defines the structure of the entire conversation and defines and routes user to the different sub dialogues;

And contains the following data class:

1. **Report details -** This data class contains the various fields according to the software structure and additionally the DocumentType field defined in Section 5.1;

Figure 6.5: Initial prototype dialogues.

In this first version of the prototype the "welcome dialog" acts as the entry-point of the bot for the user. The bot will respond with a welcome message stating that it is in-fact a bot and can help the user reporting bugs. The user can then respond with a welcome response, this response will not be processed as it is most-likely a greeting. This dialog satisfies requirement R14 and R11 as described in Section 3.5.

The welcome message is followed by the main dialog, this dialog contains the routing and conversation flow definition. As it is possible to prompt a message from this dialog directly and process the response to this message we have chosen to implement the "what" question directly in this dialog. The response to this question will be directly saved into the "ReportDetails" data object filling the "what" field. In addition, if the LUIS application successfully finds a "WhoEntity", "WhereEntity" or "DocumentTypeEntity" these will be mapped to the accompanying fields in the data object. With this information filled up we can further control the flow of the conversation in future iterations.

Finally, the bot will provide the user with a simple summary of the extracted entities and intents stored in the data object to provide the user with feedback and afterwards ends the conversation. This flow in conversation partially satisfies requirement R10 and R13 as defined in Section 3.5.

6.3.2 LUIS application design

The initial design of the LUIS application focuses on detecting and mapping the "what" intent correctly. As a LUIS application should always contain a "NoneIntent" which will be used when none of the defined intents can be detected, we have also added this intent with 15 example utterances for the none intent and 12 for the what intent, visible in Appendix F, according to the best-practices defined [4]. This results in the following intents:

1. **What Intent -** This intent is the starting point of the prototype as it triggers the "what" question as defined in Section 6.3.1.
2. **None Intent -** This intent will be taken when the LUIS application fails at mapping an answer to a defined intent and contains Section 5.1;

Additionally, we have chosen to add three entities, for the "WhoEntity" and "WhereEntity" we have chosen to use a machine-learned entity type. This choice has been made due to the fact that the "who" or "where" element in an answer might not always be the exact name or a synonym. An example that we can give is: *"When I view an ECG HiX crashes."* in this example "I" refers to the user currently using the bot, and "view" refers to the ECGViewer as the user is trying to view an ECG. These are not actual users or locations and therefore would not be mapped to a given entity, when using a pre-defined list of possible values for this entity. By using the machine-learning type of entity the application keeps on improving in detecting values like this and will eventually be more accurate than a pre-defined list. Additionally, this reduces the amount of effort in maintaining these entities and keeping them up-to-date. However, this will result in the manual tagging of the utterances to train the application to detect the correct words.

In contrast, the `DocumentTypeEntity` is designed as a pre-defined list of values and synonyms that point to these values as displayed in Figure 6.6. This choice has been made due to the fact that HiX only supports a pre-defined set of document types and this set does not change often. Furthermore, there are a lot of synonyms to these document types that are used by the users and we can now easily extract the given document type in the bot. By using this pre-defined list of synonyms we satisfy requirement R7 as defined in 2.1. This results in the following list of entities in this version:

1. **Who entity -** Machine-learned entity that tries to identify and extract the "who" from the answer to the "what" question;
2. **Location entity -** Machine-learned entity that tries to identify and extract the "where" from the answer to the "what" question;
3. **Document type entity -** Pre-defined list of support document types in HiX and their synonyms as displayed in Figure 6.6.

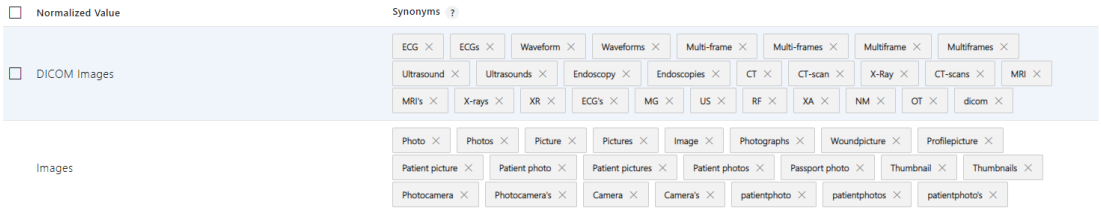


Figure 6.6: Example Document type entity values.

6.3.3 Prototype testing results

To test this version of the prototype we have chosen to use ten existing software requests available at ChipSoft. These software requests were randomly selected and two developers at ChipSoft were asked to reproduce these software requests through interaction with the prototype. The entire test set is available in Appendix E in addition the results of the tests are displayed in Figure 6.7.

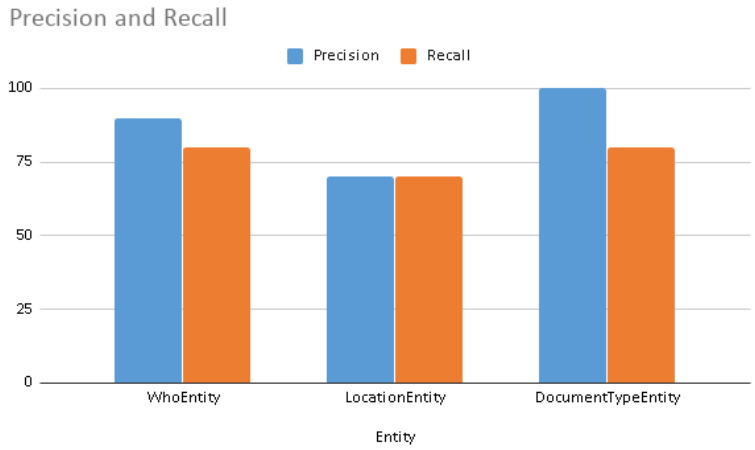


Figure 6.7: Precision & Recall per entity version 1.

As expected and due to the fact that the application had only 1 intent and the none intent in this version, the overall prediction score regarding intent prediction was 100%. In addition, we have measured the times the application failed at detecting an entity or provide us with false positives. For the testing of the application we only look at the detection of the `WhoEntity`, `DocumentTypeEntity` and `LocationEntity` due to the fact that these are available in all versions of the prototype and comparison considerations. The results of these measurements can be found in Figure 6.7. As we can see the first version of the prototype was quite accurate for the given test set as it had an overall recall of 63,33% which was calculated using the formula

$$100 - (10 * missedEntities)$$

This version had a high precision which is calculated using the following formula

$$100 - (10 * falsePositiveEntities)$$

The precision was 86,67%. The prototype had the most problems with detecting the `LocationEntities` as it missed 3 out of 10, and provided us with the same number of false positives. This can be

related to the fact that this version of the prototype detects verbs such as "open", "view" and "import" as `LocationEntities`. The reason for this is that these verbs are commonly used as nouns. These verbs themselves provide us with very little information about the location of the bug and might be combined with the `DocumentTypeEntity` in further versions to provide the developers with more information. In contrast, the application could easily detect the `DocumentTypeEntities` in the answers, this could be related to the fact that these are detected using a pre-defined list instead of the machine learning as the training set contained only 30 example utterances for the training of the model.

Finally, the utterances were manually labeled and used as extra training for the model. This way, the detection rate should increase for the next cycle of the prototype development process.

6.4 Second prototype cycle

The second prototype cycle focused on improving the accuracy of the entity tagging further developing the conversation flow and adding confirmation dialogues to validate the tagged entities. We will first go through the bot design followed by the design of the LUIS application the training of the model and test results.

6.4.1 Bot Design

This cycle focused on validating the extracted entities, adding the desired end-result dialogues and adding the "where", which refers to the location where the bug occurs, and "who", which refers to the user or user group affected by the bug, dimension follow-up questions according to the conversation flow in Appendix D The prototype contained the following **dialogues**:

1. **Welcome dialog -** This dialog states the welcome message as defined in Appendix D and helps;
2. **Main dialog -** This dialog defines the structure of the entire conversation and defines and routes user to the different sub-dialogues;
3. **Where dialog -** If the `LocationEntity`, which points out the location of the bug, can not be extracted from the answer to the "what question" defined in the main dialog, this dialog will start asking the user to describe where the problem was experienced. After processing the answer the user will be asked to validate the detected "where" value;
4. **Who dialog -** If the `WhoEntity`', which points to the users or user groups affected by the bug, can not be extracted from the answer to the "what question" defined in the main dialog this dialog will start asking the user to describe where the problem was experienced. After processing the answer the user will be asked to validate the detected "who" value;
5. **End result dialog -** This dialog records the desired end-result and puts this end-result in the already existing data object. In addition, it validates both the values for the "who" and "where" dimension residing in the data object.

And contains the following data class:

1. **Report details -** This data class contains the various fields according to the software structure and additionally the DocumentType and DesiredEndResult fields defined in Section 5.1;

As displayed in Figure 6.8 the prototype now has more control over the flow of the conversation, as it can detect if a certain entity has been tagged and if not move to a sub-question that helps it to extract that entity. In addition, the prototype can now further validate the extracted entities and move to the point where the user can define the desired end-result which in turn satisfies both requirement R1 and R2 as defined in Section 2.1.

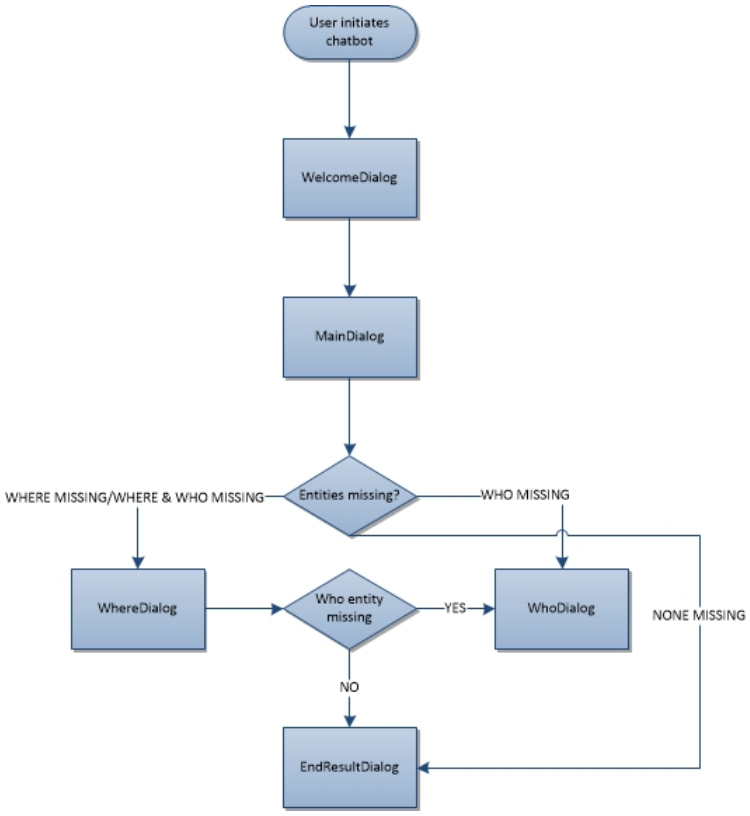


Figure 6.8: Second prototype dialogs.

By validating the entities with the user we can simultaneously provide them with feedback and insights in the choices made by the bot. For this reason we can partly satisfy requirements R10 and R11 as described in Section 3.5.

6.4.2 LUIS application design

This cycle focused on correctly labeling and training the model, adding the desired result intent and adding patterns to the what intent. The patterns were not added in the first cycle as the LUIS documentation suggests to add these in later iterations to further refine and improve the detection of the application [4]. This results in the following intents:

1. **What Intent -** This intent is the starting point of the prototype as it triggers the "what" question as defined in Appendix D.

2. **Desired result Intent -** This intent detects when the user wants to define the desired end result through the prototype as displayed in Appendix D.

3. **None Intent -** This intent will be taken when the LUIS application fails at mapping an answer to a defined intern.

In this version we have not added any entities yet as we wanted to further increase the accuracy of the entity detection as defined in Section 6.3. So the list of entities remains:

1. **Who entity -** Machine-learned entity that tries to identify and extract the "who" from the answer to the "what" question;

2. **Location entity -** Machine-learned entity that tries to identify and extract the "where" from the answer to the "what" question;
3. **Document type entity -** Pre-defined list of support document types in HiX and their synonyms as displayed in Figure 6.6.

6.4.3 Prototype testing results

To test this version of the prototype we have again chosen to use existing software requests available at ChipSoft. These software requests were randomly selected and two developer, which were different than the developers used previously, at ChipSoft were asked to described these software requests through interaction with the prototype. The entire test set is available in Appendix E in addition the results of the tests are displayed in Figure 6.9.

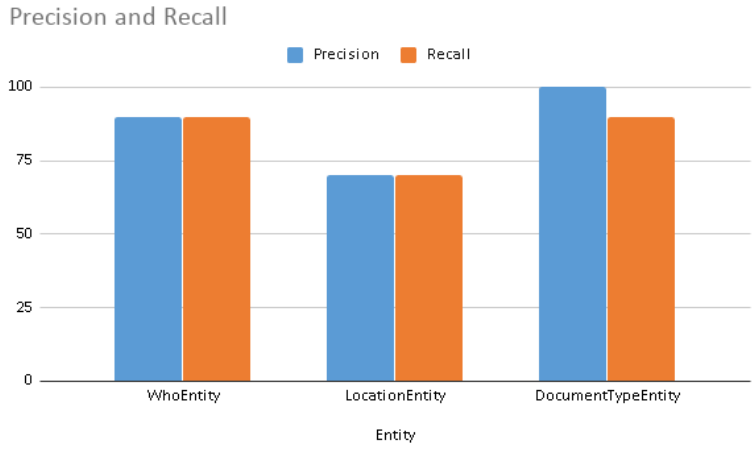


Figure 6.9: Precision & Recall per entity version 2.

As expected and due to the fact that the application had only 2 intents and the none intent in this version, the overall prediction score regarding intent prediction was 100%. For testing the accuracy and error chances of the application we have used the same testset as available in Appendix E for comparison considerations. The results of these measurements can be found in Figure 6.9. As we can see this version of the prototype had a small increase of 7% in recall in comparison to the first version. This might be a result of the fact that the model had a bigger pool of training data. In contrast, the precision remained the same and it failed at the exact same sentences as the first version did. This might be related to the fact that this version of the prototype focused on providing the application with more training data instead of adding new things to the LUIS entity detection algorithm. The prototype, again, had the most problems with detecting the `LocationEntities` as it missed 3 out of 10 and provided us with the same number of false positives. In contrast, the application had a better score at detecting the `WhoEntity` and the `DocumentTypeEntity` in the answers, this could be related to the increase in training data available to the application. This version of the application contained a total of 85 example utterances.

6.5 Third prototype cycle

The third prototype cycle focused on extending the conversation flow by adding the document type, edit report and problem result dialogues and improving the accuracy of the entity tagging.

In addition, we have added the reconstruction scenario, finalize dialog and the possibility to edit the report. We will first go through the bot design followed by the design of the LUIS application the training of the model and test results.

6.5.1 Bot Design

This cycle focused on expanding the current conversation flow by adding the document type, edit report and problem result dialog as displayed in Appendix D. The prototype contained the following **dialogues**:

1. **Welcome dialog -** This dialog states the welcome message as defined in AppendixD and helps;

2. **Main dialog -** This dialog defines the structure of the entire conversation and defines and routes user to the different sub-dialogues;

3. **Where dialog -** If the `LocationEntity` can not be extracted from the answer to the "what question" this dialog will start asking the user to describe where the problem was experienced. After processing the answer the user will be asked to validate the detected "where" value;

4. **Who dialog -** If the `WhoEntity` can not be extracted from the answer to the "what question" this dialog will start asking the user to describe where the problem was experienced. After processing the answer the user will be asked to validate the detected "who" value;

5. **End result dialog -** This dialog records the desired end-result and puts this end-result in the already existing data object. In addition, it validates both the values for the "who" and "where" dimension residing in the data object.

6. **Reconstruction scenario dialog -** This dialog records the steps taken by the user up until the point that the bug occurs, this helps the developers in the process of providing a fix for the bug;

7. **Finalize dialog -** This dialog summarizes the report and shows it to the user to provide them with feedback and add a layer of control. This helps in the process of detecting malfunctions and or errors;

8. **Edit report dialog -** This dialog will first ask the user if it wants to edit the report after the summarization. If so, the user will be provided the list of the different elements and can choose which elements it wants to edit. Afterwards, the user is redirected to the finalize dialog again and can choose to edit other elements.

And contains the following data class:

1. **Report details -** This data class contains the various fields according to the software structure and additionally the DocumentType, DesiredEndResult and ReconstructionSteps fields defined in Section 5.1;

As displayed in Figure 6.10 we have made some changes to the flow of the conversation as it seemed more logical to first validate if an entity exists and otherwise execute the dialog. In addition, we now offer the possibility to provide a reconstruction scenario step-by-step which in turn satisfies requirement R4 as defined in Section 2.1. Finally the finalize dialog adds feedback and provides extra control and error detection, this in turn satisfies requirement R6, R10 and R19 as defined in Section 2.1

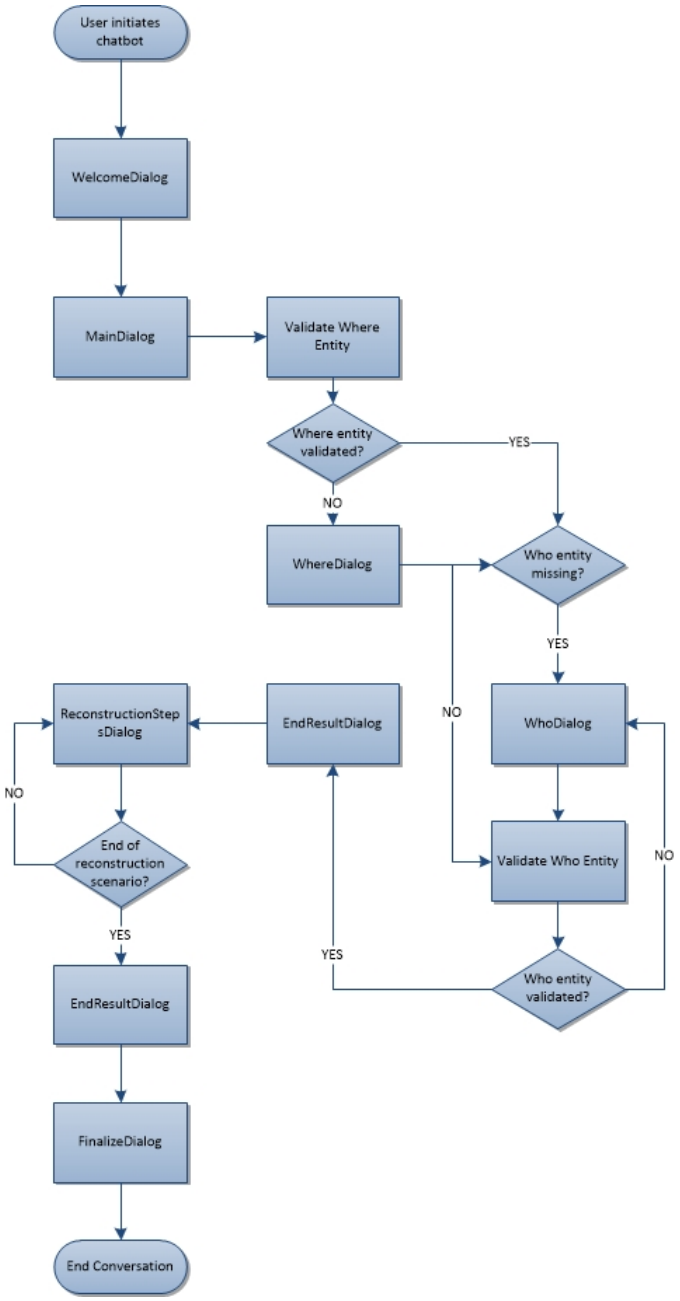


Figure 6.10: Third prototype dialogues.

6.5.2 LUIS application design

This cycle focused on correctly labeling and training the model, adding the desired result intent and adding patterns to the "what" intent. The patterns were not added in the first cycle as Microsoft suggests to add these in later iterations to further refine and improve the detection of the application [4]. This results in the following intents:

1. **What Intent -** This intent is the starting point of the prototype as it triggers the "what" question as defined in Appendix D;

2. **Desired result Intent -** This intent detects when the user wants to define the desired end

result through the prototype as displayed in Appendix D;

3. **None Intent -** This intent will be taken when the LUIS application fails at mapping an answer to a defined intent and contains.

In this version we have not added any entities yet as we wanted to further increase the accuracy of the entity detection as defined in Section 6.3. So the list of entities remains:

1. **Who entity -** Machine-learned entity that tries to identify and extract the "who" from the answer to the "what" question;
2. **Location entity -** Machine-learned entity that tries to identify and extract the "where" from the answer to the "what" question;
3. **Document type entity -** Pre-defined list of support document types in HiX and their synonyms as displayed in Figure 6.6.

6.5.3 Prototype testing results

To test this version of the prototype we have again chosen to use existing software requests available at ChipSoft. These software requests were randomly selected and two other developers at ChipSoft were asked to described these software requests through interaction with the prototype. The entire test set is available in Appendix E in addition the results of the tests are displayed in Figure 6.11.

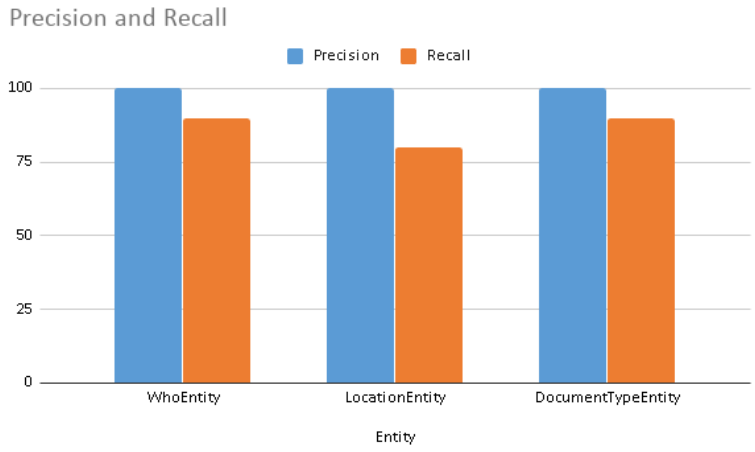


Figure 6.11: Precision & Recall per entity version 3.

In this version of the prototype, it is now possible to start with the description of the desired end result right after the welcome message dialog and the bot will correctly detect it and adjust the flow of the conversation.

For testing the precision and recall of the application we have used the same test set as available in E for comparison considerations. The results of these measurements can be found in Figure 6.11. As we can see this version of the prototype had a small increase of roughly 3% in recall in comparison to the previous version. This might be a result of the fact that the model had a bigger pool of training data. In addition, the increase mostly comes forward out of the fact that this version seemed to be better in detecting correction `LocationEntities`. In contrast, the precision

is increased to an absolute maximum of 100%, on the dataset used, which means that this version did not provide us with false positives. As in the previous versions, this version struggled with detecting every `LocationEntity` as well. The detection of the `DocumentTypeEntity` and `WhoEntity` remained the same in comparison to the previous version. This version of the application contained a total of 110 example utterances.

6.6 Fourth prototype cycle

The fourth and final prototype cycle focused on finishing up the conversation flow by adding the document type, edit report and problem result dialogues and improving the accuracy of the entity tagging. In addition, we have added an "action" field by combining the document type and where entity to provide a way to detect the action the user was performing. Finally, we have added the storing of the final result in a text file and provided keyword detection to generalize the locations and users as "i" and "we" can refer to the current user using the chatbot. We will first go through the bot design followed by the design of the LUIS application the training of the model and test results.

6.6.1 Bot Design

This cycle focused on expanding the current conversation flow by adding the document type, edit report and problem result dialog as displayed in Appendix D The prototype contained the following **dialogues**:

1. **Welcome dialog -** This dialog states the welcome message as defined in Appendix ... and helps;
2. **Main dialog -** This dialog defines the structure of the entire conversation and defines and routes user to the different sub-dialogues;
3. **Problem result dialog -** This dialog offers the users the possibility to select what the bug results in. It consists of the following choices: "Crash", "Missing functionality", "Functionality not working" and "Long waiting times";
4. **Where dialog -** If the "where entity" can not be extracted from the answer to the "what question" this dialog will start asking the user to describe where the problem was experienced. After processing the answer the user will be asked to validate the detected "where" value;
5. **Document type dialog -** If the "document type entity" can not be extracted from the answer to the "what question" this dialog will start asking the user to describe what document type is related to the problem. The list of document types can be extracted directly from the software application and is currently a static list;
6. **Who dialog -** If the "who entity" can not be extracted from the answer to the "what question" this dialog will start asking the user to describe where the problem was experienced. After processing the answer the user will be asked to validate the detected "who" value;
7. **End result dialog -** This dialog records the desired end-result and puts this end-result in the already existing data object. In addition, it validates both the values for the "who" and "where" dimension residing in the data object.
8. **Reconstruction scenario dialog -** This dialog records the steps taken by the user up until the point that the bug occurs, this helps the developers in the process of providing a fix for the bug;
9. **Finalize dialog -** This dialog summarizes the report and shows it to the user to provide them with feedback and add a layer of control. This helps in the process of detecting malfunctions and or errors;

10. **Edit report dialog -** This dialog will first ask the user if it wants to edit the report after providing the report summary. If so, the user will be provided the list of the different elements and can choose which elements it wants to edit. Afterwards, the user is redirected to the finalize dialog again and can choose to edit other elements.

And contains the following data class:

1. **Report details -** This data class contains the various fields according to the software structure fields defined in Section 5.1. Additionally, the DocumentType, DesiredEndResult, ReconstructionSteps, Action and ProblemResult fields have been added.

As displayed in Figure 6.12 we have made some changes to the flow of the conversation as it seemed more logical to first detect what the bug results in and then move on to the "where" followed by a validation or definition of the document type related to the bug. This may provide the developers with a better understanding of the bug at hand and in turn can further pinpoint the location and origin of the bug.

6.6.2 LUIS application design

This cycle focused on correctly labeling and training the model, adding problem cause entity, the combined location document type entity, this location document entity combines a location with a document type found and results in the action that the user took when the bug occurred. This results in the following intents:

1. **What Intent -** This intent is the starting point of the prototype as it triggers the "what" question as defined in Appendix D;

2. **Desired result Intent -** This intent detects when the user wants to define the desired end result through the prototype as displayed in Appendix D;

3. **None Intent -** This intent will be taken when the LUIS application fails at mapping an answer to a defined intent and contains.

In this version we have added the `ProblemResultEntity` and the `LocationDocumentEntity`. The problem result entity is used to identify what happens when the error occurs so we can further define the priority of the given error. The combination of the location and document entity can help in identifying the action the user was performing when the error occurred. And may provide the developers with a better understanding of the location of the bug:

1. **Who entity -** Machine-learned entity that tries to identify and extract the "who" from the answer to the "what" question;

2. **Location entity -** Machine-learned entity that tries to identify and extract the "where" from the answer to the "what" question;

3. **Document type entity -** Pre-defined list of support document types in HiX and their synonyms as displayed in Figure 6.6;

4. **Problem result entity -** Machine-learned entity that tries to uncover what the problem results in e.g. an error or a functionality not working;

5. **Location document entity -** Machine-learned composite entity consisting of both a Document type and Location entity that tries to identify and extract the "where" in combination with the document type from the answer to the "what" question;

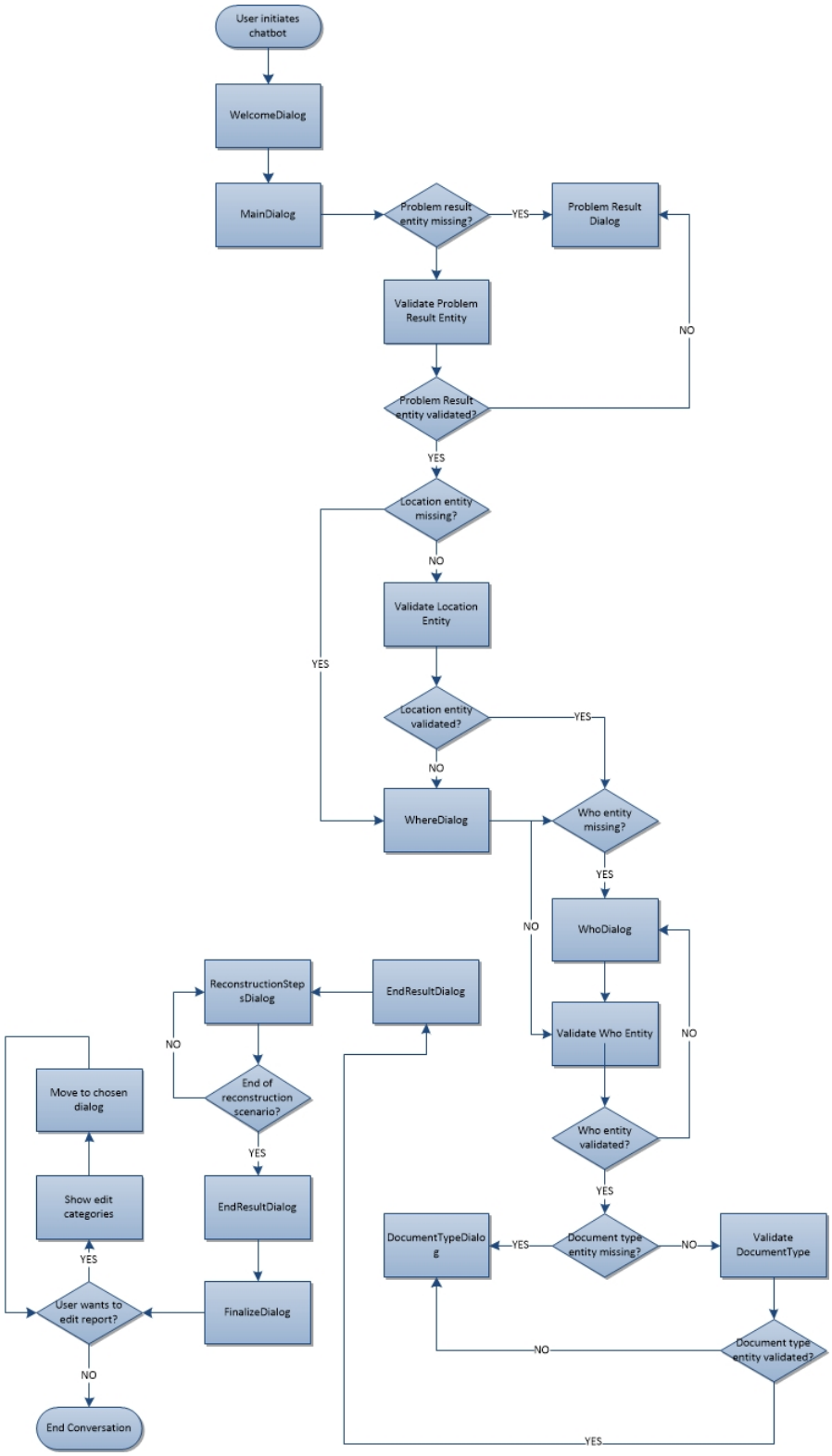


Figure 6.12: Fourth prototype dialogues.

6.6.3 Prototype testing results

To test this version of the prototype we have again chosen to use existing software requests available at ChipSoft. These software requests were randomly selected and two other developers at ChipSoft were asked to described these software requests through interaction with the prototype. The entire test set is available in Appendix E in addition the results of the tests are displayed in Figure 6.13.

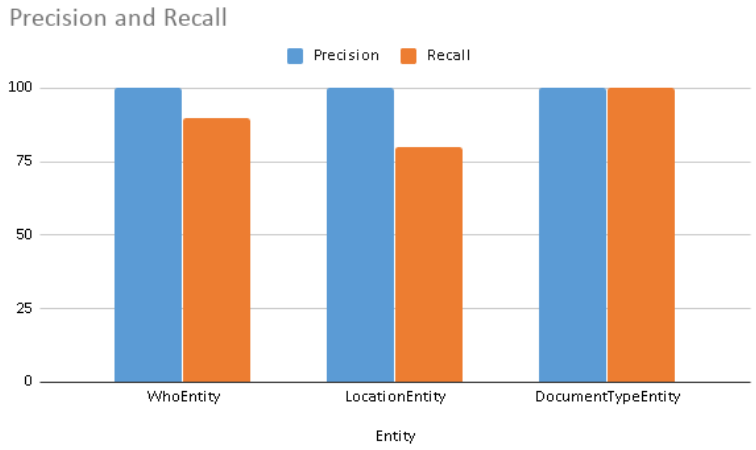


Figure 6.13: Precision and Recall per entity version 4.

For testing the recall and precision of the application we have used the same test set as available in E for comparison considerations. The results of these measurements can be found in Figure 6.13. As we can see this version of the prototype had a small increase of roughly 3.5% in recall in comparison to the previous version. This might be a result of the fact that the model had a bigger pool of training data. In addition, the increase mostly comes forward out of the fact that this version seemed to be better in detecting correction `DocumentTypeEntities`. Again, the precision was 100% which means that this version did not provide us with false positives in any way on the data that we have used. As in the previous versions, this version struggled with detecting every `LocationEntity` as well. The detection of the `DocumentTypeEntity` and `WhoEntity` remained the same in comparison to the previous version. This version of the application contained a total of 120 example utterances.

6.6.4 Proto-Requirements

By having defined all the fields that we will be extracting and using in the prototype we can define the end-result of the generated report. Therefore we can now specify the proto-requirements that we have discussed earlier in this research. The fields that will be available in the end-report and are thus referred to as proto-requirements are displayed in Figure 6.14.

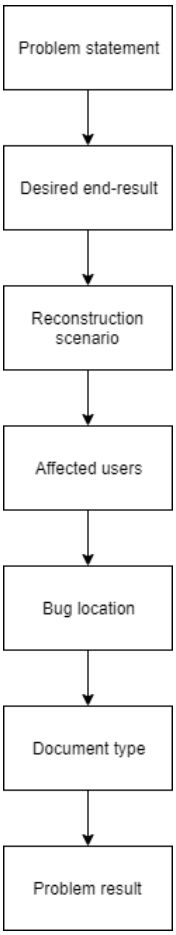


Figure 6.14: Proto-requirements definition

6.7 Machine learning analysis

With the current prototype versions and the limited amount of data we had available within the case organization, we can look at an estimation of the amount of data that was needed to train the LUIS application to the point where it could prove to be valuable in a prototype environment.

Due to time and security constraints we have chosen not to implement the prototype in the software application as it would add little to no value to the research. This means that implementing the chatbot would only prove that it is possible to implement the chatbot in an application, rather than proving the value of implementing a chatbot in an application. This means that if the prototype is to be implemented in the case application, it would use information available in the system such as patient numbers, type of desktop and user ID's.

The prototype was developed in four iterations over the research period, each iteration contained a design, development, testing and training phase. For each version a different amount of training data was fed to the system to increase the prototypes recall and precision. The amount of training data, or example utterances, per version was:

- **Version 1 -** 30 example utterances;
- **Version 2 -** 85 example utterances;
- **Version 3 -** 110 example utterances;

- **Version 4 -** 120 example utterances.

The number of example utterances that were added each version depended on several factors such as: available software requests and available developers that could aid in providing the prototype with data. Since there was no dataset of conversations regarding the domain available we could not use this to train the prototype, which in turn resulted in a minimally trained model and thus, lower recall and precision during the experiment phase.

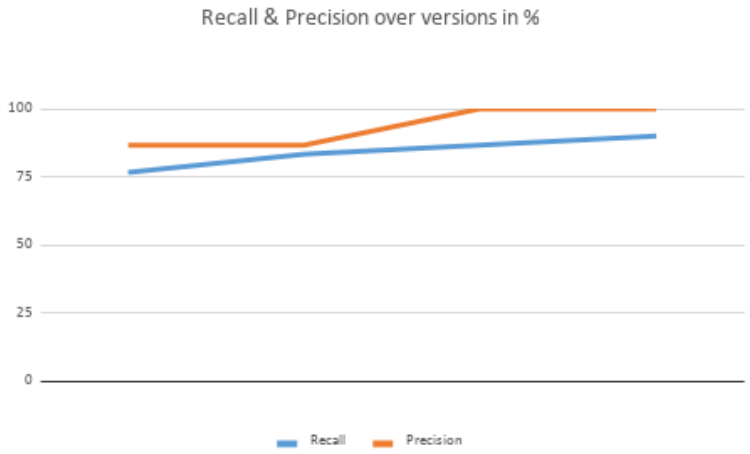


Figure 6.15: Recall and precision over versions.

As we have used the same test set to test each version of the prototype we have created a graph of the increase in recall and precision over the different versions, this is displayed in Figure 6.15. As explained and discussed in the previous sections we can see an overall increase in recall and precision over the different versions. What is interesting is the increase in precision between version 2 and 3. This increase might be the result of the increased training data or the different approach in the LUIS application as explained in Section 6.5. Another minimal increase can be noticed in recall when looking at versions 3 and 4, this might be a result from introducing the "location document entity" as explained in Section 6.6. We can notice that adding a lot of example utterances does not necessarily means increasing recall and precision by the same amount.

Chapter 7

Prototype evaluation

In this Chapter, experiments that were performed to evaluate the design of the prototype in Chapter 6 will be explained.

7.1 Experimental design

The experiment consists of a group of 3 out of 6 available Implementation & Support consultants of team multimedia. These consultants implement HiX at the customers, test new software additions and provide support for the customers. In addition, we have also included 3 out of a total of 7 developers from team multimedia. This makes the total sample size of the experiments 6 where there are 3 developers and 3 consultants. The developers that were included in the experiment have not been involved in the training process of the prototype in the previous stages of the research.

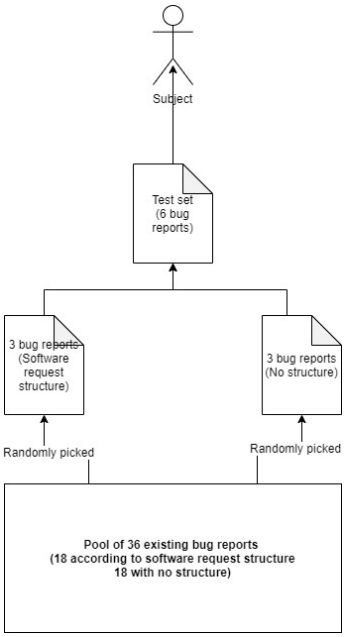


Figure 7.1: Test set assignment.

Each member of the sample group was asked to report a bug through the prototype by using existing software requests available in the current system. Each consultant reported 3 software requests conform the software request structure as explained in Section 5.1 and 3 software requests

that are not in line with the software request structure, thus contain less information. Each subject was assigned to a randomly picked test set of software requests using a numbering for each test set ranging from 1 up to 6 as we are using 6 pools of software requests. Each test set will exclusively contain bug reports and will not contain feature requests. This process is displayed in Figure 7.1.

Each of the subjects got a short introduction to the purpose of the chatbot and a small demonstration on how the systems works. We have observed the full process and took note of the interaction between the subject and the prototype and were available to provide the subject with support during the process. The focus on this part of the experiment is on the interaction between the subject and the chatbot and can also provide us with data to further analyze the efficiency of the prototype and evaluate this with the developers afterwards. By efficiency we mean the ability of the prototype to guide the users in the bug reporting process and it's ability to automatically detect entities.

When the subjects have finished reporting the pool of software requests assigned to them, we have conducted a semi-structured interview. The protocol for this interview can be found in Appendix G. This interview mainly focused on the interaction between the subject and the chatbot. However, it contained a number of questions about the correctness of the automatically detected entities and the opinion of the subject about using the chatbot as a supporting tool for the purpose of requirements engineering. The full conversation between the chatbot and user were saved for analysis purposes. Finally, a selection of the results, 2 conform the structure and 2 not conform the structure per subject, of the reports provided through the prototype were discussed by using the interview protocol in Appendix H with 3 different developers of multimedia to conclude if the bugs reported through the chatbot are more helpful than the reports available in the current system.

7.2 Hypotheses

To validate if the experiments were successful or not we have created a number of hypotheses:

1. **H1 -** Subjects prefer interacting with a chatbot when reporting bugs over reporting bugs through the current reporting system available at ChipSoft;
2. **H2 -** Developers value the reports generated through the chatbot higher than the reports, in line with the software request structure described in Section 5.1, available in the current reporting system available at ChipSoft;
3. **H3 -** Developers value the reports generated through the chatbot higher than the reports, not conform the software request structure, available in the current reporting system available at ChipSoft.

These hypotheses focus at uncovering the opinions of the subjects regarding the prototype and additionally validating the usefulness of the reports generated by the prototype. H2 and H3 have been created to help in answering RQ 3 as described in Chapter 4. In addition, H1 focuses on the ability to answer RQ 2 in combination with the semi-structured interview. This experiment can uncover the opinion of the subjects about reporting bugs by using a chatbot.

7.3 Result analysis

In this section the results of the semi-structured interviews with the subjects are discussed and the evaluation of the average System Usability Scale (SUS) score is reported on. As mentioned in Section 7.1 a total of 6 subjects (3 consultants and 3 developers) were asked to report 6 bugs (3 conform the software request structure and 3 not conform the software request structure) through the use of the chatbot prototype. The sample consisted of 50% male and 50% female subjects, 2 out of 3 consultants are female and 1 out of 3 developers is female. The subjects age ranged

between 24 and 50 years old. The highest completed level of education of 3 subjects was a master's degree, 2 subjects have a bachelor's degree and 1 subject has an intermediate vocational education. Out of the sample 4 subjects have worked at ChipSoft between 1 and 3 years, 1 subjects has been working at ChipSoft for 5 years and 1 subject has been working at ChipSoft for 19 years.

7.3.1 SUS score analysis

The SUS scale was used as an evaluation tool after the subjects had an experience with the chatbot. The scale tries to measure the opinion of the subject about the usability of the chatbot and results in a 0 to 100 score [6]. As we wanted to measure the global usability of the chatbot prototy we have chosen to use the SUS. The reason for this is that the SUS was constructed to "cover a variety of aspects of system usability, such as the need for support, training, and complexity, and thus have a high level of face validity for measuring usability of a system. " [6]

Out of the full sample, 3 out of 3 consultants rated the chatbot with a total score of 70 points on the SUS scale. 2 out of 3 developers rated the chatbot with a total score of 90 points and 1 developer rated the chatbot with a total score of 80 points the . This results in an average SUS score of 78.3%. In contrast the scale does not focus specifically on chatbots. In addition, since the experiment was executed using the Microsoft Bot Framework Emulator, we do not know if the look and feel of this emulator relates to the score. To mitigate this risk we have included some questions that might support this score in the semi-structured interview as displayed in Appendix G. It seemed that the first question: "I think that I would like to use this system frequently" in the SUS scored the lowest average amongst all subjects, with an average score of 2.5. 2 out of 3 consultants scored this question with a score of 2 and 1 out of 3 developers scored this question with a score of 2. Furthermore, question 4: "I think that I would need the support of a technical person to be able to use this system", in the SUS scored the highest average score with an average of 3.66 points.

7.3.2 Prototype evaluation

In addition to asking the subjects to score the chatbot using the SUS as explained in Section 7.3, we have conducted a semi-structured interview with the subjects. In this section we will go over each of the questions and discuss the results per question. The complete interview protocol can be found in Appendix G.

Question 1 - How did you like using the chatbot to report bugs?

The first questioned focused on extracting the opinions of the subjects regarding the usage of the chatbot to report bugs.

4 out of 6 subjects (all developers and 1 consultant) stated that they experienced using the chatbot as pleasant due to the structure it offered them when reporting a bug. 2 (1 developer and 1 consultant) out of 6 subjects stated that they liked using the chatbot due to the simplicity of reporting a bug through the chatbot. They liked the guidance the chatbot offered them as they were led through the reporting process. Finally, a single consultant stated that he liked the speed in reporting a bug the chatbot offered him.

Consultant 3 stated that there were some options missing in the lists of options offered at some questions (e.g. document types) and that she was not entirely sure that she had described the problem sufficiently. She stated that adding a feedback mechanism, where the user gets feedback if the information provided is sufficient would prove useful. Developer 3 explicitly stated that reporting a bug through the chatbot seemed easier than the current process and offered more structure.

Question 2 - What could be improved in the chatbot?

This questions tried identifying possible improvement points in the chatbot.

3 out of 6 subjects (1 developer and 2 consultants) stated that they would like to have more options in the list of options and improving the accuracy of the entity detection. 2 subjects (2 developers) stated that they would like a back button to edit the previous step immediately. 1 consultant would like a clarification of the "who" question. Finally, a consultant stated that adding a possibility to upload screenshots might prove useful and valuable for the developers.

Consultant 2 expected that, due to the simplicity and speed of reporting a bug through the chatbot, it might overload the consultants with bug reports. The suggestion was to provide a select number of end-users with the rights to use the chatbot for bug reporting purposes. Developer 2 suggested creating a real-time summary of the report and visualizing this to the user.

Question 3 - What did you think was the best part of the chatbot?

This question tried identifying the strengths of the chatbot.

5 out of 6 subjects (all developers and 2 consultants) stated that they thought the biggest strength of the chatbot was the structure it offered and the guidance of the user through the bug reporting system. 1 consultant, liked the entity detection algorithm and the possibility to select a specific document type over the structure.

Consultant 2 stated that the speed of reporting a bug through the chatbot was a strength.

Question 4 - What did you think about the flow of the conversation?

This question focused on identifying the strengths and weaknesses of the conversation flow. Additionally, it tried to identify the feeling of the conversation flow.

6 out of 6 subjects stated that they thought the conversation flow was good and clear. They stated that the chatbot asks the correct questions and the conversation feels natural to them. Consultant 1 suggested a more dynamic conversation when reporting more complex bugs and consultant 3 suggested changing the order of the questions. She suggested moving the reconstruction scenario after the what question and moving the desired result question after the reconstruction scenario.

Question 5 - Were the questions formulated understandable? If no, what would you improve?

This question focused on identifying the weaknesses in the conversation flow and suggestions about mitigating these weaknesses.

3 out of 6 subjects (2 developers and 1 consultant) stated that the questions were understandable for them. Consultant 3 stated that, due to the fact the order of the questions is consistent, users might not read the questions anymore. The rest of the subjects stated that some questions could use some clarification. Consultant 1 stated that the "where" question was unclear to her, consultant 2 stated that she had trouble understanding the "who" question. Finally, developer 1 suggested further clarifying the reconstruction scenario question.

Question 6 - Were there missing questions? If so, please state the missing questions?

This question focused on identifying the weaknesses in the conversation flow and suggestions about mitigating these weaknesses.

4 out of 6 subjects (3 developers and 1 consultant) agreed that there were no missing questions in the conversation flow of the chatbot. Consultant 1 stated that, adding a question questioning the number of the hotfix the user is on might prove useful. In addition, consultant 2 suggested adding a question regarding the module the user is in could prove valuable.

Consultant 3 suggested adding the possibility to add multiple testcases to the report to further specify the bugreport. In contrast, developer 1 stated that adding more questions might compromise the structure of the results which mitigates the value of the chatbot.

Question 7 - What did you think of the entity detection algorithm?

This question focused on identifying the opinion of the subjects regarding the entity detection algorithm.

4 out of 6 subjects (1 developer and 3 consultants) experienced the suggestions the chatbot made and the entities detected as pleasant. However, they agreed that the algorithm was a bit inaccurate at some points and suggested that adding more training data may increase the accuracy. 1 developer stated that the automatic entity detection would only prove useful if it is more accurate, and 1 consultant already liked the accuracy and the suggestions the chatbot made.

Consultant 2 suggested still showing the alternative options when a suggestion is made by the chatbot. It was unclear for here what she could choose from and therefore could not accurately confirm the suggestion made.

Question 8 - How did you experience the result of the prototype?

This question focused on identifying the opinion of the subjects regarding the report generated by the chatbot.

4 out of 6 subjects (all developers and 1 consultant) experienced the report as clear and handy and could understand the generated result. 2 consultants stated that a better presentation (e.g. bold fonts, headings, colors) would further clarify the reports.

Question 9 - What did you think about the way the result was represented?

This question focused on identifying the opinion of the subjects regarding the presentation of the report generated by the chatbot.

4 out of 6 subjects (1 developer and all consultants) agreed that adding headings or make the different parts of the report stand out more would benefit the user. 2 developers stated that they liked the way the report was presented and would not change anything.

Consultant 2 suggested creating the option to hide the reconstruction from the summary at the end of the conversation. She suggested using a button to show the reconstruction scenario, to avoid the summary being too big.

Question 10 - If we turn the chatbot into a working service would you use it to report bugs?

This question focused on the willingness of users to use the chatbot to report bugs.

4 out of 6 subjects (1 developer and all consultants) agreed that they would use the chatbot to report bugs only if it is integrated in HiX. 1 developer wanted to use the chatbot to report bugs even if it is not integrated in HiX, and 1 developers stated that from a developers' point of view

she would not use the chatbot to report bugs. However, she stated that look from a consultants' point of view or end-users point of view she would use the chatbot.

Consultant 3 added that if it offers the possibility to record a reconstruction scenario, it would prove even more useful. Developer 1 suggested that the chatbot should open when a bug occurs to offer the possibility to immediately report the bug. Finally, consultant 1 explicitly stated that he would not use the chatbot if it was a stand-alone application and required an extra action to open.

Question 11 - What features would you like to add to the prototype?

This question focused on extracting suggestions for future features to increase the success of a chatbot as a bug reporting platform.

6 out of 6 subjects agreed that adding the possibility to record the reconstruction scenario would be a valuable feature. In addition, 3 out of 6 subjects (2 consultants and 1 developer) stated that utilizing information residing within the application would be a helpful addition to the chatbot. 2 developers stated that adding a back button to the chatbot would be useful to edit the report on the fly.

Developer 1 added, that utilizing the usergroups to determine the knowledge level of the user reporting the bug might prove useful.

7.3.3 Prototype evaluation summary

When evaluating the results of the interviews, we have identified the following benefits of using a chatbot to report bugs:

- **Structure -** The chatbot provides the users with a useful structure in both the reporting process and the generated result;
- **Speed -** The chatbot increases the speed of reporting bugs by simplifying the process;
- **Simplicity -** By guiding the user through the process of reporting a bug the process can be simplified;

The subjects however state that the entity detection algorithm could use some more training data to increase the accuracy of the tagged entities. Another thing to keep in mind is that the formulation of the questions is an important aspect as it could interfere with the effectiveness of the report. Additionally, adding the possibility to record a reconstruction scenario is something the subjects highly desired, and the representation of the results should be improved by highlighting the different parts according to the consultants. In contrast, the developers like the structure and clarity of the generated reports. Furthermore, the list of options should be filled with all available options, this can however be mitigated by using the information residing in HiX. In addition, the users would like to further utilize the information available in HiX. Finally, 5 out of 6 subjects stated that they would use the chatbot to report bugs when it is turned in a working service and implemented within HiX.

Bug report comparison analysis

To validate and evaluate the results generated by the chatbot we have conducted a semi-structured interview with 3 developers. These 3 developers have been asked to compare 6 original software requests (3 conform structure and 3 not conform structure) with 6 bug reports generated by the chatbot resulting from the prototype evaluation. We have made sure that the developers did not get to evaluate a software request from the set they were asked to evaluate. In this section we will go over each of the questions and discuss the results per question. The complete interview protocol can be found in Appendix H.

Subject	Generated report grade	Original report grade
D1	8	7.5
D2	8.5	7
D3	7.5	6.5

Table 7.1: Scoring of chatbot per subject

Question 1 - How would you rate the report created by the prototype in comparison to the software original software request?

This question focused on rating the reports generated by the chatbot in comparison to the original software requests. We have used a 1-10 scale where 1 is the lowest score and 10 is the highest score.

Overall, all the subjects rated the generated report higher or equal to the original software requests. Developer 2 rated the generated report with the highest score of 8.5 and the original report with a 7. Developer 3 rated the generated report with the lowest score of a 7.5, and scored the original request with a 6.5. The average score of the generated report was an 8 and the average score of the original request was 7.2. The results can be found in Table 7.1.

Question 2 - Which elements score better in the prototypes result? Please explain why.

This question focused on identifying the strengths of the generated report.

The subjects all agreed that the generated reports had a better structure than the original software requests. In addition, there was less "noise" in the generated reports and they liked the addition of a reconstruction scenario.

Question 3 - What could have been better in the prototypes result? Please explain why.

This question focused on identifying the strengths of the generated report.

The subjects all agreed that the generated reports had a better structure than the original software requests. In addition, there was less "noise" in the generated reports and they liked the addition of a reconstruction scenario.

Question 4 - Which report do you prefer? Please explain why.

This question focused on identifying the preferences of the subjects regarding the generated report.

All subjects state that they would prefer the generated report over the original software request. Developer 1 and 2 agree that the structure and the smaller amount of noise residing in the generated report was the reason why they preferred it over the original software request. Developer 1 even stated that he thought "the structure is the biggest strength of the chatbot".

Question 4 - Which report do you prefer? Please explain why.

This question focused on identifying the preferences of the subjects regarding the generated report.

All subjects state that they would prefer the generated report over the original software request. Developer 1 and 2 agree that the structure and the smaller amount of noise residing in the generated report was the reason why they preferred it over the original software request. Developer 1 even stated that he thought "the structure is the biggest strength of the chatbot".

Question 5 - Do you think the report generated by the prototype is in line with the software request structure?

This question focused on identifying if the generated report is in line the software request structure as described in Section 5.1.

It seemed unclear to the subjects what the software request structure was. Therefore, all of them agreed that they were not familiar with it and could not rate it in comparison to the software request structure. This can be seen as a result and supports the statement that there is still a lot of structure missing in the software requests.

Question 6 - Do you think the report contains sufficient information for you to fix the bug?

This question focused on evaluating if the generated report contains enough information for the developers to fix the bug.

All developers agreed that the generated report contains enough information for them to be able to fix the bug at hand. However, these bug reports were built from bug fixes they had handled in the past.

Generated report evaluation summary

With an average score of 8 for the generated report and an average score of 7.2 for the original software requests, we can say that the generated reports are a slight improvement. The subjects agree that the structure and lesser amount of noise are the biggest benefits of the generated reports in comparison to the original software requests. One developer stated that from a developers' point of view the affected users might not be necessarily helpful to them. In addition, the software request structure defined by ChipSoft as described in Section 5.1 seems unfamiliar to the developers which supports the statement of the missing structure. Finally, all the subjects agree on the fact that the generated report contains sufficient information for them to fix the bug at hand.

Chapter 8

Conclusions & Discussion

In this section the different findings and results of this thesis are discussed and concluded. Main findings of this research are given and each research question is answered separately. Additionally, the limitations and threats of this thesis are discussed and directions for future work are proposed.

8.1 Conclusion

8.1.1 Research questions

The research questions, as proposed in the first chapter, are as follows:

1. **RQ 1 -** How can users be motivated to provide feedback through a chatbot integrated into a software product?
2. **RQ 2 -** How can chatbots help guiding users in providing informative feedback about the software product?
3. **RQ 3 -** How can proto-requirements be automatically elicited from natural-language user feedback?

The research questions are answered in the subsections in this chapter. After these answers, the lesson learned for the main research question is discussed. The main research question was stated as follows: MRQ. *"How can chatbots integrated within a software product prove beneficial in eliciting requirements from user feedback?"*

8.1.2 Motivating users to provide feedback through a chatbot integrated in a software product (RQ 1)

The insights gathered in Section 3.2 and 3.3 show that involving users in the software development process is a crucial yet hard process. It is clear that we are aiming for intrinsic motivation as this focuses on getting users genuinely interested in the software product and willing to commit time in the software evolution. Using semi-structured interviews with the consultants and developers of the case company. We have identified the different aspects that influence this willingness, and have turned them into requirements for the chatbot prototype described in Section 3.5.

Due to security and time constraints we were not able to implement the chatbot into HiX. This automatically resulted in a small number of opinions. In addition, the sample contains only subjects with a technical background. This could mean that actual end-users could have a harder time interacting with the chatbot. Due to this threat and constraints we were not able to test what moves end-users to using the chatbot for bug reporting purposes.

8.1.3 Guiding users in providing informative feedback about a software product using a chatbots (RQ 2)

In this thesis we have created a prototype of a chatbot. Additionally, we have created conversation flows, specific to the case company, resulting from discussion with the first supervisor of this thesis and the chatbot expert at the case company. Finally, we have added a machine-learning solution to the prototype that can automatically detect entities by the answers provided by the user. However, we have identified that for the process of bug reporting, the 5W's as explained in Section 5 can be used as a red thread to guide the users in providing sufficient information for the developers to fix the bug. In addition, the automatic entity detection could provide the users with suggestions to ease the process of bug reporting.

These 2 principles could prove applicable to situations other than the case companies'. Furthermore, the subjects agreed that utilizing the information residing in the application would be valuable. Finally, the interviewees stated that adding the possibility to record a reconstruction scenario would be a very helpful addition.

8.1.4 Automatically eliciting proto-requirements from user feedback (RQ 3)

By adding the machine-learned entity detection algorithm, we can automatically detect entities in the answers given by the users. In addition, by summarizing and creating a visual bug report we are able to create a bug report that is equivalent to proto-requirements. The proto-requirements, that were used in this research are defined by the fields available in the bug reports of the case company and defined in the software request structure as displayed in Figure 6.14 and Section 5.1. The interviewees in this thesis have stated that they liked the structure the reports offered them and the reduction in the noise in comparison to the original software requests. In addition, the subjects experienced the suggestions made by the chatbot as pleasant but stated that more training data might improve the accuracy. The suggestion was to improve the presentation of the reports by adding headings or bold fonts to further clarify the different elements.

8.1.5 Using chatbots integrated in a software application to elicit requirements

We had a sample of 6 subjects (3 developers and 3 consultants) available. Additionally, the bug reports used in the experiment were already available at the case company and no new bugs could be reported through the chatbot. Based on the results we can conclude that at the given case study the subjects stated that they would use the prototype when it is implemented in the software application to report bugs through. They have also rated the bug reports generated by the chatbot higher than the original existing software requests. Most of the subjects agree that adding an option to record a reconstruction scenario could prove useful in the future and further improving the presentation of the bug report might clarify it even more for the developers. Finally, by combining all the facets of the chatbot such as the automatic entity detection, ability to guide users and utilize the information residing in the software application it could prove to be a useful requirements elicitation medium.

8.2 Lessons learned

We have learned that, when correctly trained and implemented, a chatbot could prove to be a valuable requirement elicitation medium. The users seemed to like the structure the chatbot offered and the guidance through the process of reporting bugs. Furthermore, the amount of training data was limited resulting in a lower detection rate of the entities. Once there is more data available the accuracy of the entity tagging process may be increased mitigating this problem.

Another thing is that for bug reporting the 5W's can provide a red thread to both guide the users in the conversation flow and provide structure to the developers in the reports.

8.3 Validity threats

In this section the validity threats regarding the research will be explained and mitigated.

The first and foremost validity threat is the limited sample size, resulting in a small number of opinions and the fact that the sample used in this research only contained subjects with a technical background. This could mean that actual end-users could have a harder time interacting with the chatbot. This threat was introduced due to the fact that it is simply not possible to implement the chatbot in HiX and let end-users interact with the chatbot due to security and time constraints. However, by asking the subjects to act and use the chatbot like an end-user would and additionally asking them to think from the role of an end-user in the interviews could mitigate this threat. Moreover, the consultants are well capable of looking at the chatbot from and end-users point of view as they are working and communicating with end-users on a daily basis.

The second validity threat could be the limited amount of training data the chatbot had available. This could result in the chatbot missing out entities or even returning false positives, which could affect the opinions of the subjects about the chatbot. This threat is mitigated by providing the subjects with feedback during the reporting process and giving them the opportunity to edit and correct automatically detected entities.

Due to the exploratory nature of this research it was not possible to collect large amounts of data. Therefore, the lack of quantitative data could prove a threat to the validity of this research. However, it is not the goal of this research to prove a given algorithm or technology, the goal is to prove the value of a chatbot implemented in a software application as a requirements extraction method. To prove this we have used the opinions of domain experts in combination with requirements and chatbot experts available to prove this, thus possibly mitigating this validity threat.

Another validity threat could be the limited amount of available software requests. This could result in the use of software requests that were used during the training of the prototype as there are simply no new software requests available within ChipSoft. This could pose a threat as the chatbot could have an easier time recognizing the entities from these requests. We have tried to mitigate this threat by randomly selecting the software requests and have only used the most recent available software requests. However, the possibility of selecting a software request that was used during the training process still exists.

The final threat could be the fact that the chatbot used during the experiments was not implemented inside the software application. Therefore, it was not possible to simulate the full experience of a chatbot integrated within a software application. Therefore, relying on the subjects' creativity to recognize the possibility of extracting and using data residing inside the application. We have tried to mitigate this risk by explaining the possibilities of implementing the chatbot in the application during the semi-structured interviews executed after the experiments. This way the subjects could recognize the possibilities of implementing the chatbot in HiX without the need to implement the chatbot in HiX.

8.4 Future work

Due to the exploratory nature of this resource of the combination of CrowdRE, User feedback and chatbots it is only a fundament for the scientific opportunities and optimization of this field of Requirements Engineering. In this section we propose future work to further analyze and research

this field of Requirements engineering.

8.4.1 Analysis of automatic entity detection opportunities

Since this research had a limited amount of participants and training data available, a following research could focus on delving deeper into the different automatic entity methods. One can think of the pattern-matching algorithms, NLP & Machine-learning. In addition, the researchers could try to gather more training data and examples to test the different approaches and propose a best practice for this specific case of automatic entity detection in the field of CrowdRE. Furthermore, they could quantitatively analyze these approaches to provide scientifically solid proof of the best fitting approach.

8.4.2 End-user analysis

While this research has focused on identifying the needs and wishes of a specific group of involved end-users, a future research might include end-users. This might lead to interesting results as motivating end-users can still prove to be a hard activity. Specifically identifying different ways to intrinsically motivate end-users to use a chatbot to provide feedback over a given software application. In addition, involving end-users might lead to interesting flaws in the conversation flows and eventually the automatic entity detection algorithms.

8.4.3 Implementing the chatbot in a software application

As described we were not able to implement the chatbot in an actual software application and have only proposed ideas on how implementing the chatbot might prove useful. For this reason, a next research could actually delve deeper into the possibilities and opportunities of implementing the chatbot in a software application. One might think of utilizing the information available within the software application as proposed in this research.

Bibliography

[1] luis (language understanding) cognitive services microsoft azure. 44

[2] Raian Ali, Carlos Solis, Mazeiar Salehie, Inah Omoronyia, Bashar Nuseibeh, and Walid Maalej. Social sensing: when users become monitors. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 476–479. ACM, 2011. 10

[3] Muneera Bano and Didar Zowghi. User involvement in software development and system success: a systematic literature review. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, pages 125–130. ACM, 2013. 11

[4] Dina Berry. Best practices - luis - azure cognitive services, Jul 2019. 51, 54, 57

[5] Johanna Bragge, Hilkka Merisalo-Rantanen, and Petri Hallikainen. Gathering innovative end-user feedback for continuous development of information systems: a repeatable and transferable e-collaboration process. *IEEE Transactions on Professional Communication*, 48(1):55–67, 2005. 11

[6] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996. 67

[7] Robert Dale. The return of the chatbots. *Natural Language Engineering*, 22(5):811–817, 2016. 1, 14, 15

[8] Leela Damodaran. User involvement in the systems design process-a practical guide for users. *Behaviour & information technology*, 15(6):363–377, 1996. 11

[9] V Dheepa, D John Aravindhar, and C Vijayalakshmi. A novel method for large scale requirement elicitation. *International Journal of Engineering and Innovative Technology (IJEIT) Volume*, 2, 2013. 1, 10

[10] Jeremy Dick, Elizabeth Hull, and Ken Jackson. *Requirements engineering*. Springer, 2017. 6

[11] Dylanavalverde. A brief history of chatbots, Dec 2018. 18

[12] Edwin Friesen, Frederik Simon Bäumer, and Michaela Geierhos. Cordula: Software requirements extraction utilizing chatbot as communication interface. In *REFSQ Workshops*, 2018. vivi, 15, 16

[13] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1276–1284. ACM, 2013. 11

[14] Futurism. The history of chatbots [infographic], Nov 2016. 18

[15] Eduard C Groen, Joerg Doerr, and Sebastian Adam. Towards crowd-based requirements engineering a research preview. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 247–253. Springer, 2015. 1

[16] Eduard C Groen and Matthias Koch. How requirements engineering can benefit from crowds. *Requirements Engineering Magazine*, 2016. 8, 10

[17] Eduard C Groen, Norbert Seyff, Raian Ali, Fabiano Dalpiaz, Joerg Doerr, Emitza Guzman, Mahmood Hosseini, Jordi Marco, Marc Oriol, Anna Perini, et al. The crowd in requirements engineering: The landscape and challenges. *IEEE software*, 34(2):44–52, 2017. vivi, 1, 8, 9, 10, 17

[18] Jonathan Grudin. Interactive systems: Bridging the gaps between developers and users. *Computer*, (4):59–69, 1991. 11

[19] Geoff Hart. The five w's: An old tool for the new task of task analysis. *Technical communication*, 43(2):139–145, 1996. 40

[20] Richard Harwell, Erik Aslaksen, Roy Mengot, Ivy Hooks, and Ken Ptack. What is a requirement? In *INCOSE International Symposium*, volume 3, pages 17–24. Wiley Online Library, 1993. 6

[21] Jennifer Hill, W Randolph Ford, and Ingrid G Farreras. Real conversations with artificial intelligence: A comparison between human–human online conversations and human–chatbot conversations. *Computers in Human Behavior*, 49:245–250, 2015. 31

[22] Mahmood Hosseini, Keith Phalp, Jacqui Taylor, and Raian Ali. The four pillars of crowdsourcing: A reference model. In *Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on*, pages 1–12. IEEE, 2014. 9

[23] Mahmoud Hosseini, Keith T Phalp, Jacqui Taylor, and Raian Ali. Towards crowdsourcing for requirements engineering. 2014. 10

[24] Netta Iivari. Enculturation of user involvement in software development organizations-an interpretive case study in the product development context. In *Proceedings of the third Nordic conference on Human-computer interaction*, pages 287–296. ACM, 2004. 11

[25] Max Kush. The statement problem. *Quality Progress*, 48(6):71, 2015. 40

[26] Carlene Lebeuf, Margaret-Anne Storey, and Alexey Zagalsky. Software bots. *IEEE Software*, 35(1):18–23, 2018. 1, 15, 16, 17

[27] Carlene R Lebeuf. *A taxonomy of software bots: towards a deeper understanding of software bot characteristics*. PhD thesis, 2018. vivi, vivi, vivi, vivi, vivi, vivi, vivi, vivi, 13, 14, 15, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31

[28] Chris Lindstrom. How to write a problem statement, Apr 2011. 40

[29] Philipp Lombriser, Fabiano Dalpiaz, Garm Lucassen, and Sjaak Brinkkemper. Gamified requirements engineering: model and experimentation. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 171–187. Springer, 2016. 9

[30] Walid Maalej and Hadeer Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd international requirements engineering conference (RE)*, pages 116–125. IEEE, 2015. viivii, 12, 13

[31] Walid Maalej, Maleknaz Nayebi, Timo Johann, and Guenther Ruhe. Toward data-driven requirements engineering. *IEEE Software*, 33(1):48–54, 2016. 9, 10, 12

[32] Frauke Paetsch, Armin Eberlein, and Frank Maurer. Requirements engineering and agile software development. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*, pages 308–313. IEEE, 2003. 6, 7, 8

[33] Dennis Pagano and Walid Maalej. User feedback in the appstore: An empirical study. In *Requirements Engineering Conference (RE), 2013 21st IEEE International*, pages 125–134. IEEE, 2013. 11, 12

[34] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *Software maintenance and evolution (ICSME), 2015 IEEE international conference on*, pages 281–290. IEEE, 2015. 11, 12

[35] Klaus Pohl. The three dimensions of requirements engineering. In *International Conference on Advanced Information Systems Engineering*, pages 275–292. Springer, 1993. vivi, 35

[36] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016. 14, 18

[37] Bayan Abu Shawar and Eric Atwell. Chatbots: are they really useful? In *Ldv forum*, volume 22, pages 29–49, 2007. 1, 13, 14

[38] Joseph Sirosh. Luis.ai: Automated machine learning for custom language understanding, Feb 2018. vivi, vivi, 47, 48

[39] Remco Snijders, Fabiano Dalpiaz, Mahmood Hosseini, Alimohammad Shahri, and Raian Ali. Crowd-centric requirements engineering. In *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pages 614–615. IEEE, 2014. 1, 8, 9

[40] Ian Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition, 2010. 6

[41] Melanie Stade, Marc Oriol, Oscar Cabrera, Farnaz Fotrousi, Ronnie Schaniel, Norbert Seyff, and Oleg Schmidt. Providing a user forum is not enough: first experiences of a software company with crowdre. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 164–169. IEEE, 2017. 1, 10

[42] John A Taylor. How bots work - bot service, May 2019. vivi, vivi, 48, 49

[43] Richard H. Thayer, Sidney C. Bailin, and M. Dorfman. *Software Requirements Engineerings, 2Nd Edition*. IEEE Computer Society Press, Los Alamitos, CA, USA, 2nd edition, 1997. 6

[44] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley Publishing, 1st edition, 2009. 6, 7, 8

[45] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014. vivi, 36

[46] Wizu and Wizu. A visual history of chatbots, Feb 2018. 18

[47] Eric SK Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of ISRE'97: 3rd IEEE International Symposium on Requirements Engineering*, pages 226–235. IEEE, 1997. 3

[48] Wlodek Zadrozny, Malgorzata Budzikowska, J Chai, Nanda Kambhatla, Sylvie Levesque, and Nicolas Nicolov. Natural language dialogue for personalized interaction. *Communications of the ACM*, 43(8):116–116, 2000. 1

[49] Didar Zowghi and Chad Coulin. Requirements elicitation: A survey of techniques, approaches, and tools. In *Engineering and managing software requirements*, pages 19–46. Springer, 2005. 6, 8, 11

Appendix A

Interview protocol

I&S Interview Protocol

The goal of this thesis project is to uncover how the feedback of large heterogeneous groups of users can be continuously included in the requirements engineering process, by using chatbots integrated in the software product, in this case HiX. This can help companies involve the users and provide them with a continuous stream of valuable feedback to be used in the requirements engineering process.

ChipSoft can use the feedback to elicit requirements and tailor the software product to the users, in our case the consultants, needs more accurately. Additionally, the conversations between the chatbot and the users can be analyzed to improve the understanding of the users' needs and wishes, resulting in a more successful software product and greater user satisfaction.

The chatbot focuses on gathering feedback from a large heterogeneous group of users to elicit feedback from these requirements. It should provide the consultants with a medium to report bugs and request features from within HiX. The chatbot will have two clear functionalities:

1. Requesting a new feature;
2. Reporting a bug.

Furthermore, the chatbot should use the feedback to elicit requirements from through a natural-language conversation. It is important that the chatbot focuses on the "why" dimension rather than the "what" dimension.

This interview is executed to gather more insights in the current processes and elicit requirements from the stakeholders, in this case the consultants, for the prototype.

Interviewee information

1. What is your name?
2. What is your age?
3. What is your gender?
4. What is your education level?
5. How long have you been working at ChipSoft?
6. What is your job at ChipSoft?

Process questions

First off, we will start with questions to clarify the process of bug reporting and feature requests and how a feedback chatbot can provide support in this process.

1. Can you explain the current process of requesting a feature for the standard content in the multimedia/PACS module?
2. Can you explain the current process of reporting a bug in the standard content in the multimedia/PACS module?
    a. Is this process the same as in other modules?
3. What is your opinion about these processes?
4. Do your feature requests or bug reports get rejected at times?
    a. What is the reason behind this?
    b. Do you think further clarification of the requests/reports would decrease the rejection rate?
5. What would you do to improve these processes? Please motivate your answer.
6. Do you often get questioned to clarify your requests or reports by the developers or teamleads?
    a. If yes, how do you communicate with the developers?
    b. What is your opinion about this way of communication?
7. Can you explain/sketch a conversation to request a feature?
8. Can you explain/sketch a conversation to report a bug?

Chatbot questions

The next questions will focus on gathering your opinion about the use of a chatbot for gathering feedback and eliciting requirements through feedback in HiX.

1. How would you like to use a chatbot to provide feedback about HiX?
2. What type of feedback regarding the standard content should the chatbot be able to gather? Please motivate your answer.
3. Are there any **must-have** questions the chatbot should ask for eliciting requirements?
    a. If yes, which questions?
        i. Can you motivate why these are must-have questions?

Interaction questions

The next couple of questions focus on the way you would like to interact with the chatbot.

1. How do you think the chatbot should interact with humans and why?
    a. Image Command-driven chatbot
    b. Image natural language chatbot
2. Should the chatbot be able to handle multiple tasks simultaneously e.g. Bug reporting when requesting a feature?
    a. Can you motivate your answer?
    b. Can you provide an example of such a situation?

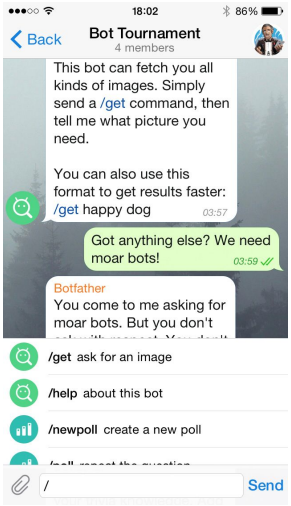
3. How would you describe the end-state of the chatbot or is there no end-state in a conversation?
4. Please explicitly define the chatbots goals.

Chatbot identity questions

The final questions will focus on the identity of the chatbot and what the identity of the chatbot should be according to you.

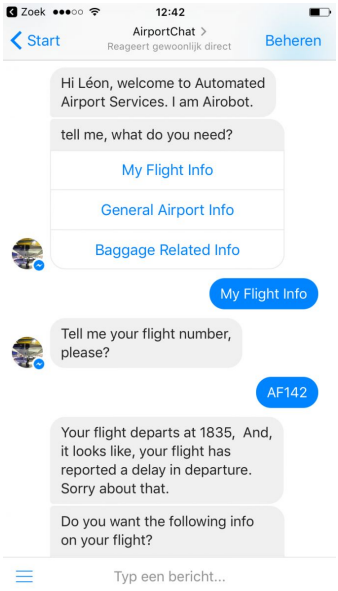
1. Do you think the chatbot should have a personality?
    a. Should the chatbot have a name?
        i. If yes, what name do you suggest?
    b. Should the chatbot have a gender?
        i. If yes, what gender would you prefer and why?
    c. Should the chatbot age over time or should it have a static age?
    d. Should the chatbot have an unique way to identify it? E.G. a logo, 2D avatar or 3D avatar.
        i. If yes, which?
    e. Should the chatbot have a visible or identifiable species, race, or ethnicity?
        i. If yes, what do you think it should be?
    f. Should the chatbot have a clear profession?
        i. If yes, what should it be?
    g. Do you think the chatbot should reply to personal questions as if it has its own personality?
        i. What personal questions should the chatbot be able to reply to?
    h. Do you think the chatbot should show emotions?
        i. If yes, in what way should the chatbot show its emotions, on a superficial level or logical level? - Plaatje
        ii. Superficial
        iii. Logical
2. Should the chatbot be able to deceive the users that it is a human? Explain why or why not.

A. **Command-driven bot**

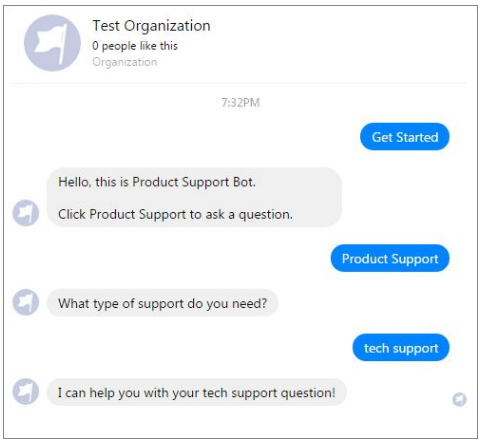


B. **Natural-language bot**

A. Proactive



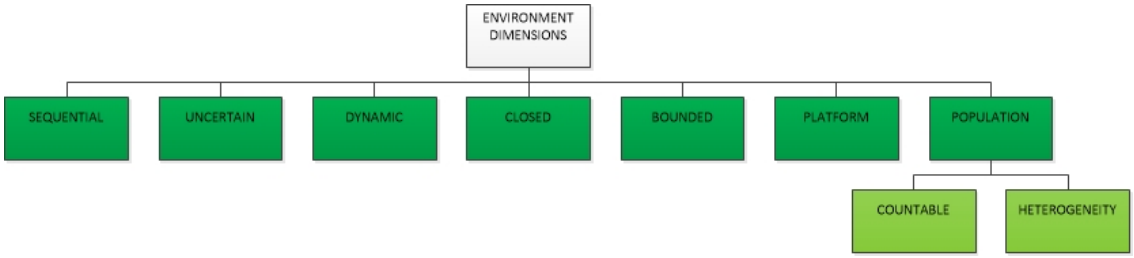
B. Reactive



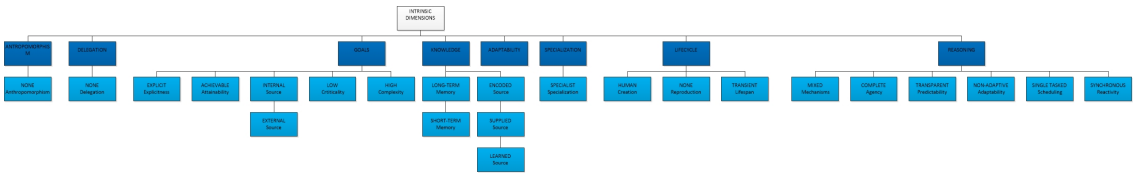
Appendix B

Software bot taxonomy

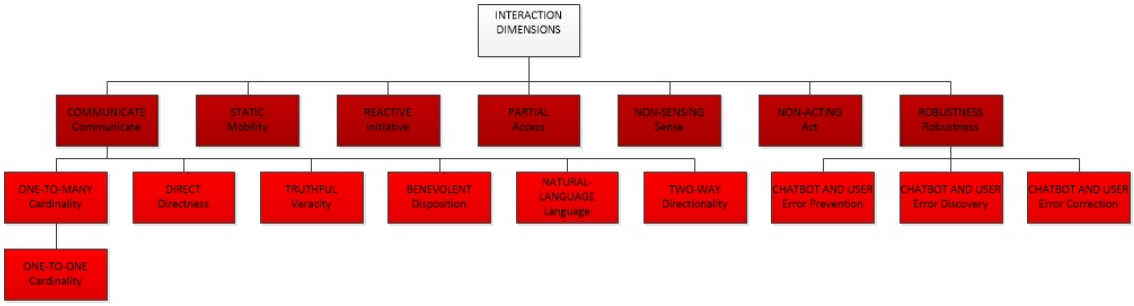
B.1 Environment dimensions



B.2 Intrinsic dimensions



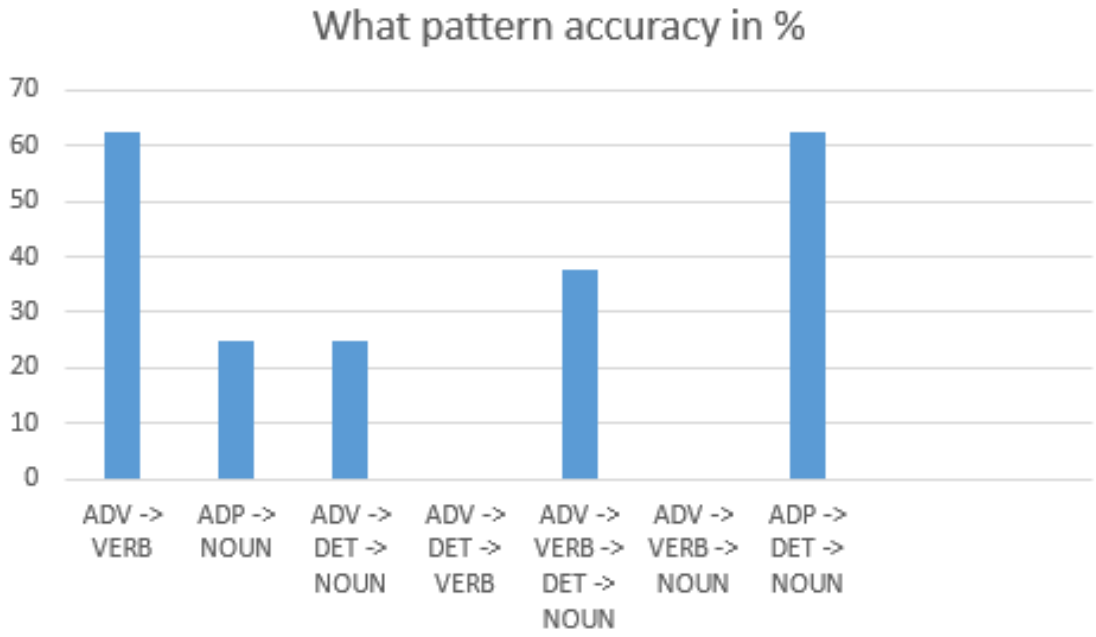
B.3 Interaction dimensions



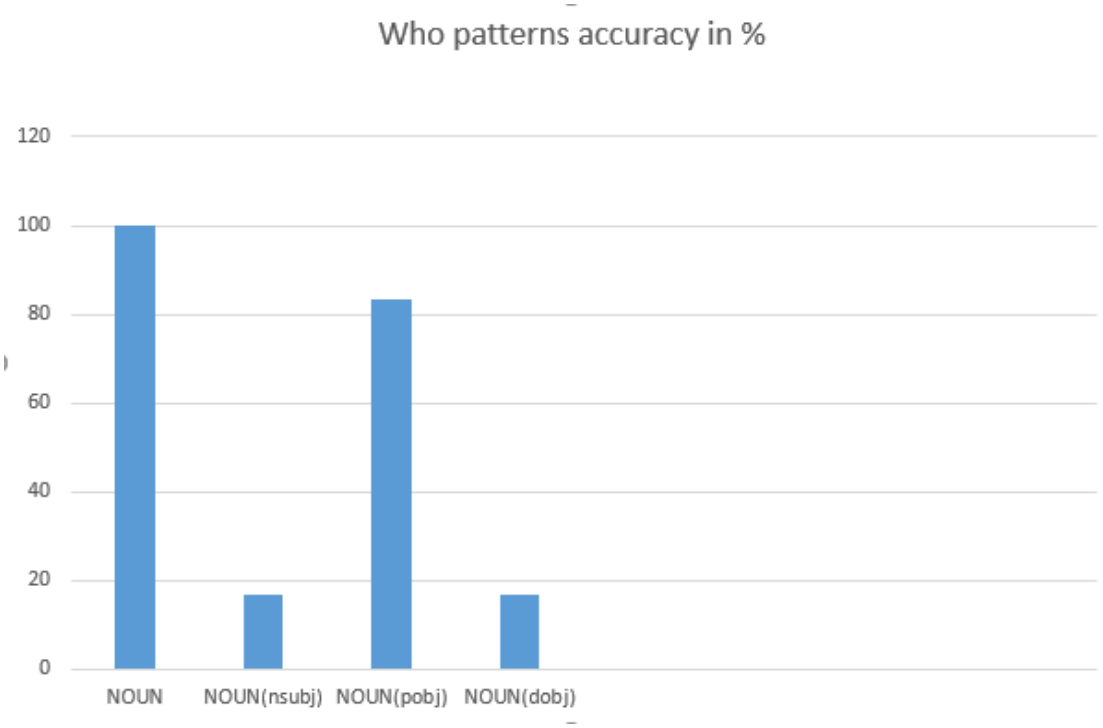
Appendix C

Pattern accuracy results

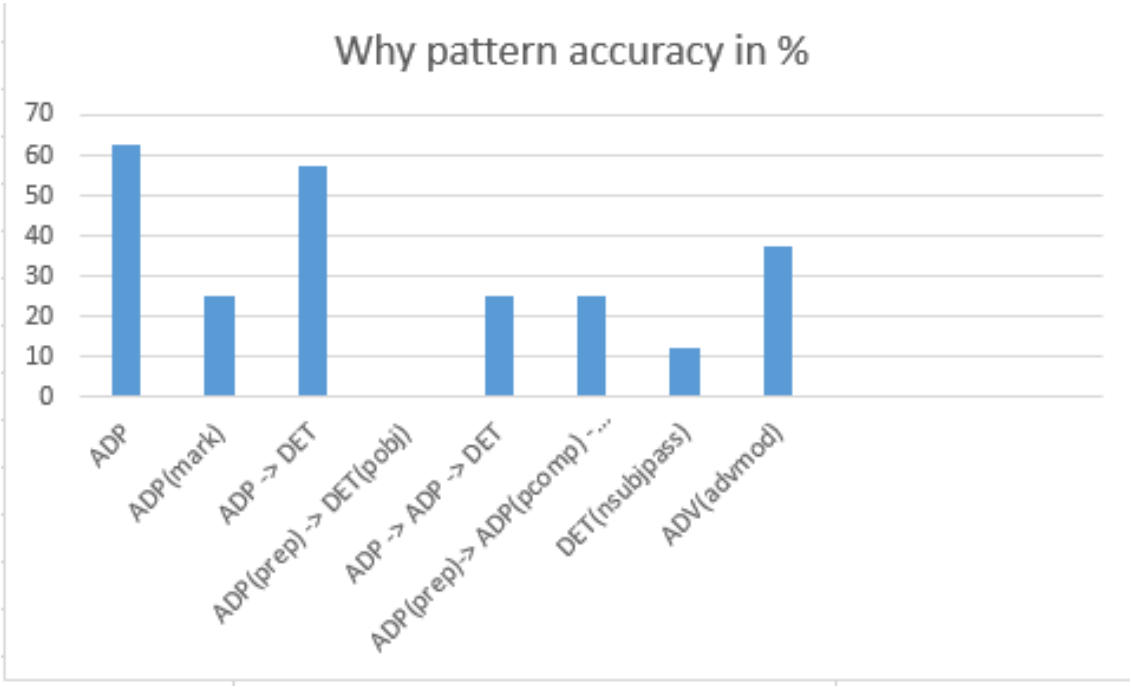
C.1 What pattern accuracy



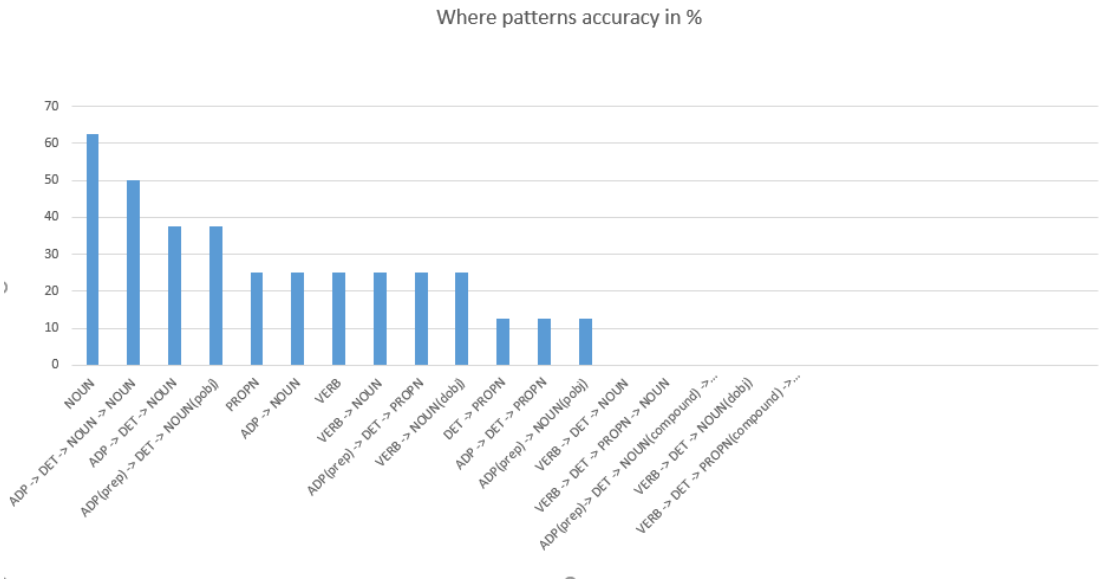
C.2 Who pattern accuracy



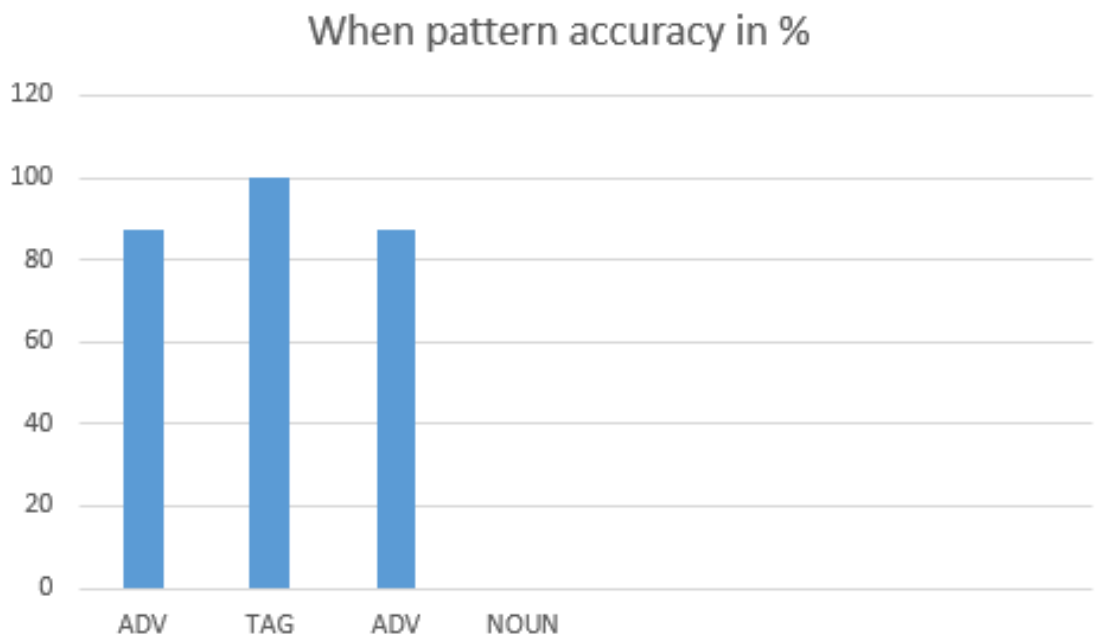
C.3 Why pattern accuracy



C.4 Where pattern accuracy



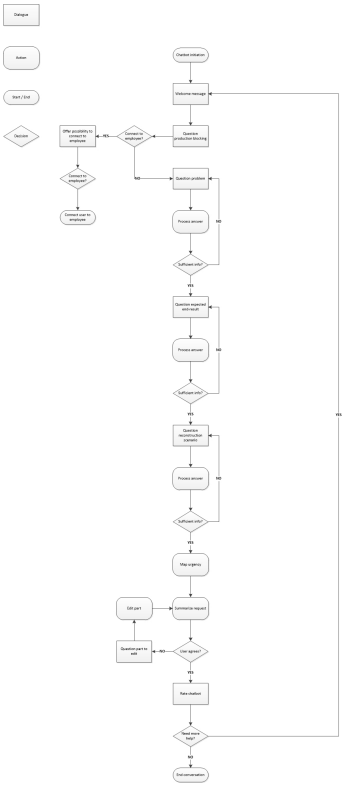
C.5 When pattern accuracy



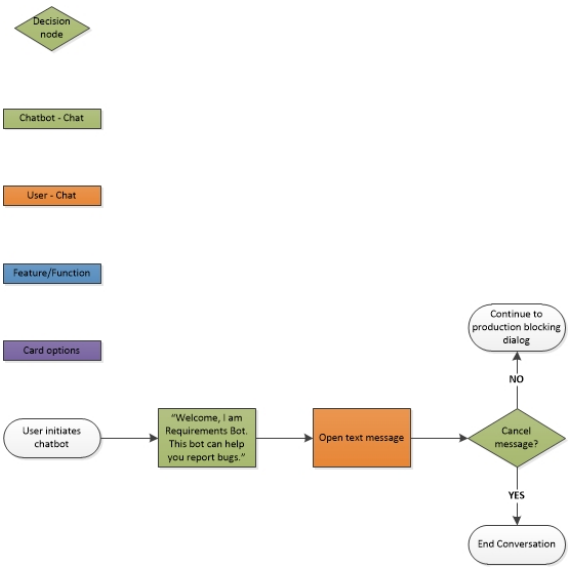
Appendix D

Initial conversation flow

D.1 Full conversation flow

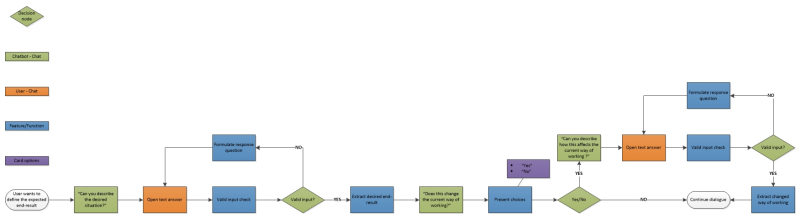


D.2 Welcome dialog

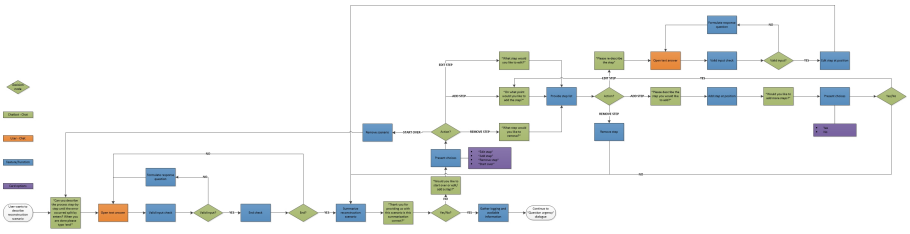


D.3 Question problem dialog

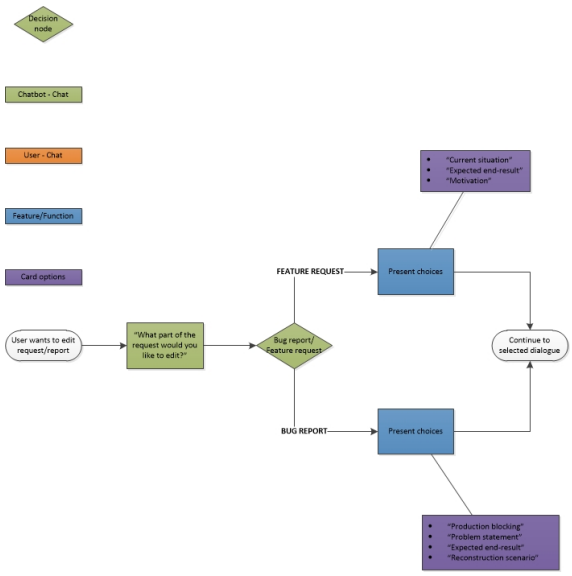
D.4 Expected end-result dialog



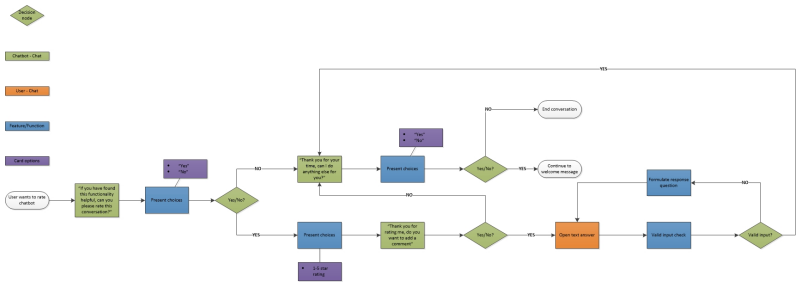
D.5 Reconstruction scenario dialog



D.6 Edit dialog



D.7 Rate chatbot dialog



Appendix E

Prototype testsets

E.1 Testset version 0.1

1 - Jip melding bij geen juiste gebruikersrechten om een drive mapping te benaderen
A Jip notification appears when in certain situations the files cannot be accessed correctly by HiX. Part of the Jip notification is a Win32 epoch utc date time adjusted to the local date time. This date is 1-1-1601 and HiX does not like that as Creation date time. This date comes from a .Net function that returns this if no valid files are found in the relevant call. With this Sif, we want to ensure that such Jip reports do not occur more often in situations like this.

2 - Jip-melding bij openen Pdf document met webbrowser in samengesteld formulier
Reported via support site Sint Maarten Kliniek.
When using the view multimedia document (Display of images) in a composite form, it happens to SMK that the PDF is loaded twice \ opened by refreshing the Multimedia document. Because the Multimedia document is refreshed, the region is refreshed as an action link and a new region is loaded. The view model property still comes in on the old screen in terms of binding, so this gives a Jip message on the web browser (cannot be delivered object).

Solution is by keeping a boolean variable or the dispose function off the screen and then when refreshing the Multimedia document, no longer refreshing the web browser.

3 - Progressbar bij bulkimport weer aanzetten
When performing bulk import, the two lower progress bars are not used because the bulk import task caused errors. This has since been repaired, but the progress bars are still off. We also want to merge some duplicated code from both bulk import functionalities.

4 - Doorsturen beelden scherm werkt nog niet op de nieuwe manier.
When images have been imported, a function has been created to be able to choose which examinations should be forwarded to an external pac.

This screen currently does everything itself in terms of structure etc. While in 6.2 many of these functionalities are centralized in the thumbnail browser itself.

With this sif we want to forward screen update to work in accordance with the new thumbnail browser. In addition, no valid dome server is not taken into account, we would like to immediately take this into account when we show a message to the user.

5 - Progressbar bij bulk Bij scannen document + OK+nieuw komt JIP-melding 'Kan bestand C:\Users\norim\AppData\Local\Temp\tmp1642.jpg niet vinden' naar vorenimport weer aanzetten
Reason:
Reported by: Testing sifs 6.2 73474

When scanning a (group) document, where you select 'Create group document' in the scan screen and fill in a group name and then click OK + new, a JIP message will appear: Can file C: \ Users \ norim \ AppData \ Cannot find Local \ Temp \ tmp1642.jpg. (JIP 7ADB9F3C). This also applies to not scanning as a group document. So generally the OK + new button.

Desired effect:
That no JIP message appears and that a new scan can be made via OK + new button after scanning a (group) document

6 - Fout bij dicom store koppeling
Situation: With ViaSana an error occurs that generates the following error message on the dicom rontgen store link: Unable to cast object or type 'System.String' to type 'ChipSoft.Publics.MultiMedia.ICS_MultiMediaStorageLocations. This error message keeps coming with every new message that is offered until the link is rebooted.

7 - Opslaan rotatie van PDF documenten
When saving PDF documents, the rotation must be saved per page, this must be done in a separate structure and not in the Multimedia document, because only 1 value fits in there now. Because everything is now scanned to PDF, creating / modifying this structure is necessary. It is not going well now with PDF documents that are imported as PDF files, usually these PDF documents have already been properly edited / created in advance.

The same also applies to a Multi-Page tiff, for this there is only 1 multimedia document and then the rotation must be saved per page. This must be stored in a separate table (production) that is linked to the multimedia document.

8 - Bij bulk export van multimediadocumenten moeten privacy en niet toonbare documenten niet gekozen worden

With bulk export to file or bulk export to printer, multimedia documents from a patient in a certain category can be selected for export. Here privacy and non-displayable documents can now also be selected and exported while this is not allowed. With this sif we want to filter out privacy and documents that cannot be displayed

9 - Dicom video en multiframes thumbnails tonen geen icoon en afspeeltijd

The thumbnails of dicom videos now get a dicom icon and with a dicom multiframe. With sif 105036, dicom videos get a picture in the thumbnail instead of an icon. With this sif we want to give the pictures a play icon and play time in the thumbnails.

In the code: create a virtual field that shows the effective playing time. This field is then used to display as playback time in the thumbnail. The effective playback time is determined by doing frametime * imagecount / 1000 (if it is in milliseconds), if the playback time is not yet filled.

10 - Volgende/vorige knop bij externe pdf geeft wit scherm

When switching between PDFs that are viewed with the external viewer, clicking on the next button will display a white screen instead of the PDF.

Margie 11-20

11 - Importeren vanaf fotocamera werkt niet icm citrix

In support call 526578, which states that reading photos from a camera does not work. Locally this goes well. But under citrix an error message appears: required capability not supported. With this sif we want to ensure that images can be imported from a camera without an error message. this concerns the action photo camera via scanning.

12 - Bij alle typen multimedia documenten mimetype zetten bij import

Since sif 83798 it is possible to add an action link of the type "show multimedia documents of investigation" to one or more mime types so that only multimedia documents of this mime type are shown. At the moment, however, the mime type is only set when importing dicom documents and importing PDF documents. So this functionality is still of limited value. With this sif it is the intention that when importing all multimedia documents the mime types are set. The most logical place to do this is in the "PrepareDocumentForPost" function of DocumentUtilities in the public-extend or the "NewMultiMediaDocumentAdvancedPrefill" function in the document logic.

13 - Bij het klikken op een MM documenten wordt PACS gestart via een actionlink in een actionlink

When CS-Pacs is started from the thumbnail browser (double-click on a document / research that is connected to a config that says 'use in CS-Pacs', a multimedia action link is started to open the document. Inside this action link is then looked at or pacs, or MM must be opened.

When this is PACS, a controller and action link from Pacs is created to open the builds. In accordance with the code guidelines you do not need to start action links in action links.

With this SIF we want to make this process in accordance with the guidelines. Either return the pacs action as a view result, or bring the check forward: when clicking on the document, you already determine which action link should be opened.

14 - Vertaalbug: static gedrag DrawingControl aanpassen

Ezis.Multimedia gives error in translation street. Cause is in this piece of code:

```
   public static readonly DependencyProperty DocumentRepresentationCurrentProperty =
        DependencyProperty.Register ("DocumentRepresentationCurrent", typeof (string), typeof
(DrawingControl),
                new UIPropertyMetadata (Resource.GetString
("Multimedia_AnnotationImageViewerContextmenu_ToggleRepresentationMMDoc_Image"));
```

Because this is static, it is executed during the translation. However, Resource.GetString cannot be used here because its implementation is in the HiX application, which is not present during translation.

(RnD manager will still want to adjust Developer and Supporter)

15 - Baseerror bij opslaan multimedia document zorgt ervoor dat het document niet opgeslagen kan worden

It is currently not possible to correct this error in a base terror that occurs during saving, and to save it again.

A temporary variable is read out in the CanCommit and if it is true, no storage is allowed.

By putting the variable into a try / finally block, the variable will always be reset and it is possible to successfully save a document after a base terror.

Appendix F

Prototype example utterances

F.1 Example utterances version 0.1

Example utterance	Score
Enter an example of what a user might say and hit Enter.	
ecgs cannot be displayed in the ecg viewer	0.99
when scanning using wia and a user is changing the scan config the ' remove files after import ' checkbox is not shown	1.00
when viewing an image an error message is shown instead the thumbnail	0.98
in the overview screen the function of removing a patient from an object is being called but this has not yet been implemented . also , too many zismuts are being created when importing an image .	1.00
when images are imported from a camera , the files cannot be deleted	0.98
when an examination is being split in 2 examinations , the original examination is set to expired . but the referralnote of the original examination still exists .	1.00
the problem is that a non dicom multimediadocument is connected to the wrong patient and it is not possible to replace and connect it to a different patient	1.00
when multimediadocuments are loaded , i have to wait a long time when the connection to the filestream can ' t be made	1.00
when the user marks groupdocument in the configuration of an import and the user only imports 1 document , still a groupdocument with 1 document is created	1.00
i can ' t copy an image to my clipboard , therefore i can ' t paste this when i use the scan functionality	1.00

Appendix G

Prototype interview protocol

Interview protocol

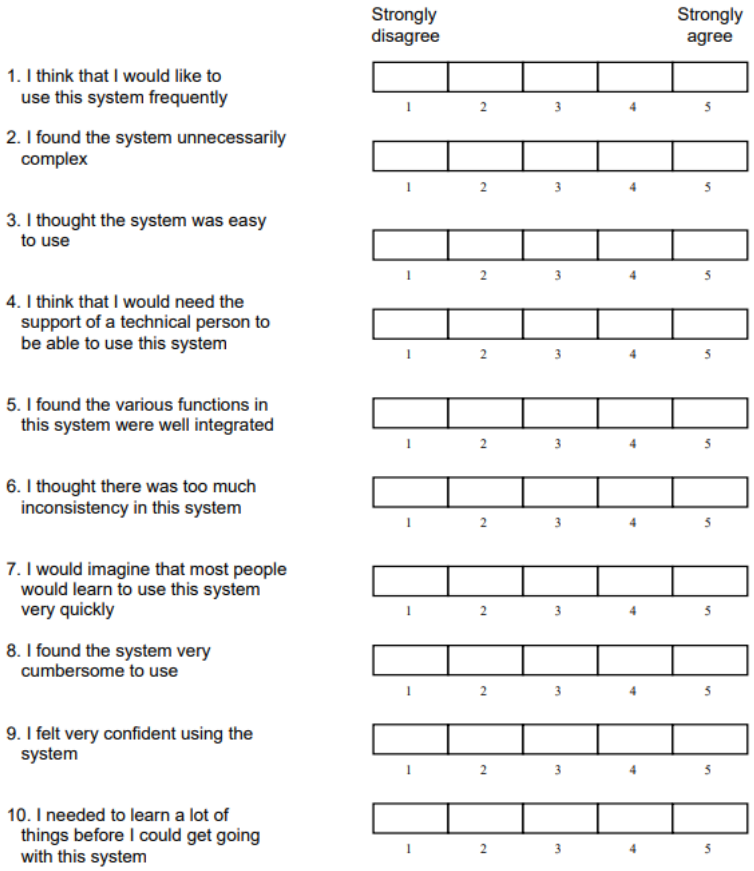
The goal of this experiment is to uncover how the feedback of large heterogeneous groups of users can be continuously included in the requirements engineering process, by using chatbots integrated in the software product, in this case HiX. This can help companies involve the users and provide them with a continuous stream of valuable feedback to be used in the requirements engineering process.

This prototype provides you with the ability to report a bug by guiding you through a certain conversation flow. This conversation flow is aimed at gathering sufficient information to create a bug report that complies with the software request guidelines available at ChipSoft.

After interacting with the chatbot, you will be asked some questions that are part of the System Usability Scale (SUS), and we will conduct a semi-structured interview to evaluate your experience of the interaction with the system. In addition, the questions will aim at extracting your opinion about the interaction with the system.

System usability scale (SUS) questions

Please provide us with some information about the usability of the scale by rating the given aspects from 1-5 where 1 is “strongly disagree” and 5 is “strongly agree”.



Prototype questions

The next questions will focus on gathering your opinion about the use of a prototype for gathering feedback and eliciting requirements through feedback in HiX.

1. How did you like using the prototype to reports bot through?
    a. What could be improved about the prototype?
    b. What did you think was the best part of the prototype?
    c. What would you improve in the prototype, and please elaborate why?
2. What did you think about the flow of the conversation?
    a. Did this flow feel natural to you? If no, what would you change?
    b. Were the questions formulated understandable? If no, what would you improve?
    c. Were there missing questions? If so, please state which you would add.
3. What did you think of the automatic entity detection algorithm?
4. Would you use the prototype in the future to report bugs? Why or why not?
5. What features would you like to add to the prototype? And why?

Appendix H

Developer interview protocol

Interview protocol-developers

The goal of this experiment is to uncover how the feedback of large heterogeneous groups of users can be continuously included in the requirements engineering process, by using chatbots integrated in the software product, in this case HiX. This can help companies involve the users and provide them with a continuous stream of valuable feedback to be used in the requirements engineering process.

This interview focuses on comparing the generated reports created by the chatbot in the experiment, with their related original software requests.

1.1 Comparison questions

1. How would you rate the report created by the prototype in comparison to the software original software request?
2. Which elements score better in the prototype's result? Please explain why.
3. What could have been better in the prototype's result? Please explain why.
4. Which report do you prefer? Please explain why.
5. Do you think the report generated by the prototype is conform the software request structure?
6. Do you think the report contains sufficient information to fix the bug reported?