



CHALMERS
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

Using machine learning and natural language processing to automatically extract information from software documentation

Master's thesis in Software Engineering

HELENA ÓLAFSDÓTTIR

MASTER'S THESIS 2019

Using machine learning and natural language
processing to automatically extract information
from software documentation

HELENA ÓLAFSDÓTTIR



UNIVERSITY OF
GOTHENBURG



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Technology
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Using machine learning and natural language processing to automatically extract
information from software documentation
HELENA ÓLAFSDÓTTIR

© HELENA ÓLAFSDÓTTIR, 2019.

Supervisor: Michel Chaudron, Department of Computer Science and Engineering
Examiner: Christian Berger, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in L^AT_EX
Gothenburg, Sweden 2019

Using machine learning and natural language processing to automatically extract information from software documentation

HELENA ÓLAFSDÓTTIR

Department of Computer Science and Engineering

Chalmers University of Technology and University of Gothenburg

Abstract

Engineers face many challenges when it comes to using and maintaining software documentation. The OD3 is a vision for the future of software documentation which proposes that documentation should be generated based on user queries. There are many steps that need to be taken to create such a system. This research takes one of those necessary steps by investigating the categories of software knowledge that are contained in software documentation, automatically classifying sentences from software documentation into those sentences, and exploring methods to identify sentence relations. This analysis was conducted on one case documentation. A system, Software Documentation Supporter (SDS), was then built to explore and evaluate the results. The aim of the SDS is to support the user when navigating through long software documentation. In the system, the user can choose from a list of questions, and the software knowledge extracted from the documentation is used to answer those questions. The results were evaluated using a quantitative and a qualitative approach. As the sample size of the evaluation was small, the quantitative results did not show a significant difference in the time it took users to solve tasks using the SDS, compared to using only the documentation. The qualitative results showed that participants did feel that the SDS supported them and that it helped them navigate the documentation, however it was also clear that improvements need to be made both in regards to the method, and the design of the system.

Keywords: software, documentation, architecture, requirement, natural language processing, classification, clustering

Acknowledgements

A large project like a Master thesis is never the work of only one person. I was lucky enough to be surrounded by people who are incredibly skillful and ambitious, and helped me achieve the results presented in this thesis. I would especially like to thank my supervisor, professor Michel R. V. Chaudron for all the time he devoted to this research. His continuous guidance and encouragement throughout the thesis was invaluable. I would also like to thank his Ph.D. assistant Rodi Jolak for his input and help with the system evaluation, and Dr. Mohamed Soliman for his insight into the subject of this thesis. I would then like to thank all the people who devoted their time to participate in the system evaluation and helped me collect valuable information for this research. Finally, I would like to thank my family and friends for their constant love and support throughout this process.

Helena Ólafsdóttir, Gothenburg, June 2019

Contents

List of Figures	xiii
List of Tables	xv
1 Introduction	3
1.1 Current practice	3
1.2 Future vision of software documentation	4
1.3 Purpose of the study	4
1.4 Problem statement	5
1.5 Scope	5
1.6 Approach	5
1.6.1 Extracting knowledge	5
1.6.2 Producing answers to user questions	6
2 Background	9
2.1 Software Documentation	9
2.2 Related Work	10
2.2.1 Similar Studies	10
2.2.1.1 SEC	10
2.2.1.2 Acquiring Architecture Knowledge for Technology Design Decisions	10
2.2.1.3 A systematic Mapping Study on Text Analysis Tech- niques in Software Architecture	10
2.2.2 Ground work	11
2.2.2.1 OD3	11
2.2.2.2 The Task Phrase Extractor	11
2.2.2.3 The Witt database	12
3 Methods	13
3.1 Research method	13
3.2 Data collection	13
3.3 Software knowledge categories and data annotation	14
3.4 Classifying sentences into software knowledge categories	14
3.4.1 Identifying relations between sentences	16
3.4.1.1 Clustering	16
3.4.1.2 Extracting and connecting sentences' task phrases . .	17
3.4.2 Identifying technology concepts	17

3.5	User questions and responses	17
3.6	Creating the Software Documentation Supporter	18
3.7	System evaluation	20
3.7.1	Participants	20
3.7.2	Usability test	20
3.7.3	User experience interview	21
3.7.4	Evaluation sessions	21
4	Design and Implementation	23
4.1	Software knowledge categories and their structure	23
4.2	Automatically extracting software knowledge	31
4.2.1	Category Classification	31
4.2.2	Identifying sentence relations	31
4.2.3	Retrieving technology concepts	32
4.3	The questions of the SDS	34
4.4	System demonstration	36
4.4.1	Q1: What functionalities exist in the system?	39
4.4.2	Q2: What functionalities does this feature provide?	39
4.4.3	Q3: How is this functionality implemented?	39
4.4.4	Q4: What was the development process related to this functionality?	47
4.4.5	Q5: What was the development process related to this non-functional requirement?	50
4.4.6	Q6: What architecture patterns are used in the system?	52
4.4.7	Q7: What programming languages are used in the system?	52
4.4.8	Q8: How are the architecture patterns in the system implemented?	53
5	Results	55
5.1	Classification results	55
5.2	Clustering results	57
5.3	Results from system evaluation	59
5.3.1	Task times	59
5.3.2	User experience	62
5.3.2.1	Results from System Usability Questionnaire	62
5.3.2.2	Results from open-ended questions	63
6	Discussion	67
6.1	Categories of software knowledge	67
6.2	Using machine learning and natural language processing to identify software knowledge	68
6.3	Usability tests	70
6.4	User experience	71
6.5	Threats to validity	73
6.5.1	Internal Validity	73
6.5.2	External Validity	75
6.5.3	Construct Validity	76

7 Conclusion	77
Bibliography	79
A Appendix A	I
B Appendix B	XIII
C Appendix C	XV
D Appendix D	XIX
D.1 Personal experience questionnaire	XIX
D.2 Usability test session	XX
D.2.1 Introduction	XX
D.2.2 Tasks to solve using documentation text	XXI
D.2.3 Tasks to solve using the Software Documentation Supporter	XXII
D.2.4 User experience interview	XXIII
E Appendix E	XXV
F Appendix F	XXIX
F.1 Results from SUS questionnaire	XXIX
F.2 Results from open-ended questions	XXIX
G Appendix G	XXXV

List of Figures

3.1	The architecture of the SDS	19
4.1	High-level view of the software knowledge category graph	25
4.2	A view of the domain category and its subcategories	29
4.3	A view of the requirement analysis category and its subcategories	29
4.4	A view of the system category and its subcategories	30
4.5	A view of the development process category and its subcategories	31
4.6	A closer view of other categories	31
4.7	An example of the results produced by the Task Phrase Extractor	32
4.8	A screenshot from the Categories table in the Witt database	33
4.9	A screenshot showing the SDS when first opened	36
4.10	A drop-down menu containing the questions that the user can ask about the documentation	37
4.11	A mock-up of the upper result box	37
4.12	A mock-up of the lower result box	38
4.13	A screenshot showing results to Q1, filtered to show only use cases for the display product feature	40
4.14	A screenshot showing the feature selector	41
4.15	A screenshot showing a part of the results to Q2	41
4.16	A screenshot showing the functional requirement selector	42
4.17	A screenshot showing results to Q3	44
4.18	A screenshot showing results to Q3-1	45
4.19	A screenshot showing results to Q3-2	46
4.20	A screenshot showing results to Q3-3	47
4.21	A screenshot showing results to Q4	49
4.22	A screenshot showing the non-functional requirement selector	50
4.23	A screenshot showing results to Q5	51
4.24	A screenshot showing results to Q6	52
4.25	A screenshot showing results to Q7	53
4.26	A screenshot showing results to Q8	54
5.1	Results from the principal component analysis	57
5.2	Manually labelled sentences	58
5.3	Clustering of feature, functional requirement and use case sentences, using the Gaussian-mixture model	60
5.4	Task times for all tasks performed in the usability testing sessions	60

5.5	A count of how often participants were faster using documentation or SDS, per task	61
5.6	A summary of what participants liked about the SDS	63
5.7	A summary of how participants felt when using the SDS	64
5.8	A summary of improvement suggestions for the SDS	65
6.1	Task times for all tasks performed in the usability testing sessions . . .	71
B.1	Clustering of feature, functional requirement and use case sentences, using the K-means model	XIII
C.1	Shapiro-Wilk and QQ-plot for documentation task times	XV
C.2	Shapiro-Wilk and QQ-plot for SDS task times	XVI
C.3	F-test that checks for equal variance in the samples	XVII
E.1	Results from the personal experience questionnaire	XXV
E.2	A bar chart showing the participants' experience with web development	XXVI
E.3	A bar chart showing the participants' experience with requirement analysis	XXVI
E.4	A bar chart showing the participants' experience with software architecture	XXVII
F.1	Results from the System Usability Scale questionnaire	XXIX
F.2	A list of comments about what participants liked about the SDS . . .	XXX
F.3	A list of comments regarding how participants felt when using the SDS	XXXI
F.4	A list of improvement suggestions from the participants	XXXII
F.5	A list of other comments received from the participants	XXXIII
G.1	Pointers given during usability testing	XXXV

List of Tables

3.1	Scikit-learn classification algorithms evaluated for each trained classifier	15
3.2	Libraries used in the system implementation	19
3.3	Evaluation session schedule	22
4.1	High-level categories of knowledge presented in software documentation	24
4.2	Labels in software documentation	24
4.3	Categories of knowledge presented in software documentation	26
4.4	The questions of the SDS	34
5.1	Classifier - hierarchy vs. flat	55
5.2	Classifier - hierarchy vs. flat	59
5.3	Results from the Welch t-test	61
5.4	SUS score per question	62
A.1	Flat approach - classifier results	II
A.2	Level 1	III
A.3	Level 2 - requirements category	IV
A.4	Level 2 - system category	V
A.5	Level 2 - development process category	VI
A.6	Level 3 - structure category	VII
A.7	Level 3 - UI design category	VIII
A.8	Level 3 - quality assurance category	IX
A.9	Level 4 - structure implementation category	X
A.10	Level 4 - behaviour implementation category	XI

Glossary

Software documentation Software documentation is a artefact of the development process of a system that has the purpose of preserving and communicating knowledge of the system under development. In this research, I will only look at the parts of the software documentation that are written in natural language.

Software knowledge category In this research, a set of categories are defined that are supposed to cover knowledge contained in software documentation. These categories are referred to as software knowledge categories.

Software Documentation Supporter (SDS) Software Documentation Supporter is the system that was developed in this research. In the system, users can view and ask questions about the documentation they are analysing. Its purpose is to support the user when navigating large documentations and help him find the desired information faster.

1

Introduction

Software documentation is an important tool used by developers to explain and justify design decisions and demonstrate the requirements, architecture and implementation of a system. These documentations are incredibly important to preserve knowledge within an organisation and to make sure information is not only tied to specific employees. They therefore serve an important role when it comes to maintaining and comprehending software systems.

Software documentation imposes various challenges in the field of software engineering. The creation, maintenance and use of these documents are all problematic and old-fashioned in the sense that they all rely on manual work. Still to this day, software developers spend vigorous amounts of time manually comprehending software architecture and implementation, for example, by reading those documents.

The idea of using technology to reduce this manual and tedious activity is not new and many advances have been made in an attempt to make software comprehension easier and less time-consuming. Most engineers and architects however still spend too much time comprehending systems and even when they only need answers to a few specific questions they still need to go through vast amounts of source code, diagrams and documentation.

It is apparent that many steps need to be taken in order to replace this old-fashioned way of software comprehension. This research will contribute to that large goal by taking one of those necessary steps.

1.1 Current practice

When developing large software, many design decisions are made that affect the source code and its structure. Those decisions are not always evident in the source code itself and must often be clarified with diagrams and justified with documentation.

Documentation of software is a very complex task and a lot of research has been conducted on the problem of preserving developers' design decisions and rationales, like for example studies from 2016 by Nosál' et al. [1] and 2018 by Luciv et al. [3] that focus on the problem of repetition in software documentation, a study from 2017 by Vranić et al. [2] that focuses on the problem of intent comprehensibility

preservation in software documentation and a study from 2015 by Robillard et al. [9] that discusses problems in API documentation. The stakeholders of those documentations are many and vary from business clients to developers maintaining the system in question. Many different views and aspects therefore need to be considered in the documentation, making it prone to duplication and misguidance. On top of this, documentation tends to have low prioritisation due to their high cost, low immediate return nature, as discussed by both Lethbridge et al. [8] and Robillard et al. [9] Developers are therefore pressured to spend little time on documenting systems, causing documentation to be incomplete and later on, poorly maintained and updated.

These facts impose extreme challenges when it comes to system comprehension. Developers often have to read through vast amounts of information that contain duplicates and are insufficient at the same time, causing them to use up to 58% of their time on system comprehension or related activities as shown by Xia et al. [7]

With developer documentation being one of the most useful information during software maintenance, insufficient documentation also makes maintaining software more time-consuming and error-prone. The fact that they often contain duplicates also makes the documentation itself very sensitive to the changes made to the system, as discussed by Luciv et al. [3]

1.2 Future vision of software documentation

The On-Demand Developer Documentation (OD3) by Robillard et al. [4] points out the high cost and low immediate return of software documentation and introduce a vision for better satisfying the information needs of software developers. The OD3 proposes a system that would generate appropriate documentation based on user queries. A user should be able to ask questions and retrieve relevant information from source code, Q&A forums, etc. Many challenges need to be overcome to create the system described. One of those challenges is retrieving various knowledge from software documentation and investigating how it can be used to satisfy the information needs of software developers. This research will focus on this particular challenge.

1.3 Purpose of the study

In order to answer questions about the content of software documentation, it is important to possess knowledge about different parts of it, such as what a specific diagram is showing and what a specific section or sentence is discussing.

The purpose of this study is to delve into the problem of extracting useful knowledge from natural language in software documentation and using that knowledge to support the user when navigating through those documents in search for specific information. The study has two main focus points. Firstly, retrieving rele-

vant information from software documentation, using natural language processing, classification and clustering algorithms, and other automatic extraction techniques. Secondly, investigating how this knowledge can be used to support users to navigate more quickly through those documentations when they are looking for specific information or answers to specific questions.

1.4 Problem statement

The following two research questions are answered in this study:

RQ2: How to automatically categorise natural language text presented in software documentation into software knowledge categories?

RQ3: How to automatically identify relations between specific instances of knowledge in software documentation?

1.5 Scope

In order to build the system envisioned in the OD3, all software artefacts and source code must be considered. This study will only consider natural language sources when extracting knowledge, eliminating all diagrams and source code related to the system in question.

The amount of training data for this research is limited, both due to difficulties to find good, open-source documentation, and due to time constraints, as all data labelling to create ground-truth had to be done manually. The data used therefore came from only one open-source software project containing 860 sentences that were labelled and used for training. The level of annotation was also limited to sentences, excluding paragraph, section and page level, again due to time constraints.

As the data is very limited, especially considering machine learning standards, deep-learning methods will not be considered in this study.

1.6 Approach

This study can be divided into two major tasks, extracting knowledge from software documentation and using this knowledge to produce sufficient responses to user queries.

1.6.1 Extracting knowledge

Understanding and comprehending natural language is a complex task. With thousands of years of experience and development, the human brain has managed to solve this challenge with great results. We are not just able to read the lines, we are able

to read between the lines, comprehending complex context, sentiment and hidden meanings. Teaching a machine to truly understand natural language is therefore an extremely complex and elusive task. In this study, two types of knowledge were extracted from the sentences; context and technology concepts. To grasp the context of the sentences, they were classified into subcategories of software knowledge. Technology concepts were identified when they appeared in the sentences. Two different methods were then used to identify relations between different sentences of the documentation.

The first step was to define the categories of software knowledge. These categories were determined using qualitative content analysis, both by looking through current literature on software knowledge and by iterating through the data provided for this research. The knowledge categories were then used to create ground truth by manually annotating each sentence of the dataset.

Features were extracted from the sentences and used for training and evaluating classifiers. For extracting the features, various algorithms were investigated and compared, such as bag-of-words and tf-idf, along with various NLP techniques, such as n-grams, lemmatisation and stemming. Finally different classifiers were trained and evaluated, using the Python machine learning package Scikit-learn.

To identify relations between sentences, two different methods were used. Unsupervised clustering was used to identify relations between features, functional requirements and use cases and a rule-based method was used to identify all other sentence relations. In the rule-based method, task phrases were first extracted from all sentences and the words of those task phrases were then compared to identify relations between different sentences of the documentation.

To identify technology concepts, a database that contains about 26,500 technology concepts was used to find matches in the documentation text.

1.6.2 Producing answers to user questions

The next step is to use the extracted knowledge to answer questions from developers or architects.

Firstly, the types of questions asked by those specialists needed to be investigated, to provide support to the most common questions. This step was carried out early and in parallel to the knowledge extraction phase, in order to make sure the knowledge needed to answer those questions is properly extracted.

Instead of using complicated methods to comprehend the user questions themselves, a list of possible questions is provided. The relevant knowledge is then retrieved from the database and presented to the user.

A system (Software Documentation Supporter (SDS)) was then developed where

users can ask questions about the documentation used for this research. Usability tests were conducted using the SDS, to evaluate the approach and results of this research.

2

Background

This section discusses the background and related work of this research. Firstly, section 2.1 briefly discusses how software documentation is used today and what challenges it faces. Secondly, section 2.2 discusses what progress is being made in regards to those challenges and where this research fits in with that, and then covers the ground work of this research.

2.1 Software Documentation

Documentation is an important part of many processes. Documentation contains information about decisions made, the rationale for those decisions, directives for future work, and other important evidence about work that has been conducted.

Object-oriented software development is one of the processes where documentation is extremely important. Different documentation is written for different steps of the development process. The most common are software requirements specification (SRS), software architecture documentation (SAD), a technical documentation, and quality assurance documentation. These documentations serve as a repository for important decisions and results, and a place for the software engineer to motivate and explain the decisions made during development. They also serve as a substitute for the engineer, since it is incredibly expensive if everyone would direct their questions straight to them or they simply might not be present anymore.

Software documentation faces many challenges in the modern world. With new technology being developed rapidly, documentation is often given low priority and very little time, resulting in low quality documentation, as discussed by Bass et al. [10]. On top of that, Lethbridge et al. show that most documentation is rarely maintained, and quickly becomes outdated [8]. Lastly, even though most software documentation is standardised by the IEEE, in practice, the engineers often improvise. When reviewing possible documentation for this research, the reality was that they are as different as they are many, both structure and content wise.

In this research, the focus will be on documentation written in natural language, thus taking into account requirements analysis, software architecture and quality assurance.

2.2 Related Work

2.2.1 Similar Studies

2.2.1.1 SEC

The SEC is a tool designed by Alex Tao and Mahsa Roodbari in 2018 [11]. The tool is designed to answer questions about requirements and architecture, based on the software documentation. It focuses on retrieving architectural information, often presented in diagrams, as well as listed features, requirements, use cases and other results from the requirement analysis. The information is manually extracted from the documentation and stored in an ontology. The ontology is then used to form answers to questions from engineers.

There are two major differences between the SEC and the SDS. Firstly the SDS will automatically extract and populate the ontology with information from documentation. Secondly, the SDS will focus on extracting knowledge from natural language text instead of knowledge contained in diagrams and lists.

2.2.1.2 Acquiring Architecture Knowledge for Technology Design Decisions

Acquiring Architecture Knowledge for Technology Design Decisions is a dissertation thesis by Mohamed Soliman from 2018 [19]. In this research, Soliman looks at architecture knowledge in Stack Overflow posts and how this knowledge can be identified and used to make better design decisions. Soliman used an ontology and a set of classification approaches to capture semantic information and to identify and classify architecture-related posts. He then implemented his approach in a web-based search engine, which proved more effective than the traditional keyword-based search.

There are two main differences between this research and Soliman's research. First, the type of data analysed and used, since Soliman analysed online posts whereas this research analyses software documentation. Second, the system produced in this research will help a user analyse one specific documentation at a time, whereas Soliman implemented his approach as a search engine for all software architecture related information gathered from Stack Overflow posts.

2.2.1.3 A systematic Mapping Study on Text Analysis Techniques in Software Architecture

This literature review from 2018 by Tang et al. [20] thoroughly covers research using different text analysis techniques in the field of software architecture, i.e. classification, clustering, search and information retrieval, etc. From this mapping study, it is apparent that many researchers focus on different ways to use text analysis

techniques to extract information that can be helpful to software architects and engineers. However, none of the papers reviewed use machine learning techniques to extract general architecture knowledge from SADs. There are three papers mentioned in this literature review that have work related to our research; Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach, from 2009 by López et al. [21] recognises the need of information extraction from SADs and has a similar idea of a system that helps users grasp specific knowledge, using information extraction and ontologies. However, their approach to extract information deviates from ours, as they use rule-based learning for this purpose. Semi-automated Design Guidance Enhancer (SADGE): A Framework for Architectural Guidance Development, from 2014 by Anvaari et al. [22] develops a framework for extracting architectural knowledge. Again however, rule-based learning is user for information extraction. Personalised architectural documentation based on stakeholders' information needs, from 2011 by Nicoletti et al. [23] attempts to identify parts of SADs that might be of interest of specific users. Their approach is to identify software architecture related concepts in SADs and use them to classify their sentences into predefined categories. Their approach is however semi-automated as they require manual semantic annotations with the SAD document, in order to produce results.

2.2.2 Ground work

2.2.2.1 OD3

The On-Demand Developer Documentation (OD3), as mentioned in section 1.2, is a vision for future software documentation. The OD3 describes a system that would replace the manual work of writing documentation, instead, the appropriate documentation is generated as an answer to a user query.

This study will contribute to the OD3 vision mainly by exploring two things.

1. What knowledge is contained in software documentation?
2. How can we extract the knowledge necessary to answer questions about specific software documentation?

2.2.2.2 The Task Phrase Extractor

Extracting Development Tasks to Navigate Software Documentation is a paper from 2015 by Treude et al. [30] In this research, a system was developed that extracts task phrases from software documentation and uses them to help the user navigate the documentation. This is done by first running a documentation through the system to identify and extract all task phrases. Those results are then used in the search functionality of the system to auto-complete and create suggestions based on what the user searched for.

The paper shows that extracted task phrases are more useful than extracted concepts and code elements. In this research, I use the Task Phrase Extractor to extract task phrases from all sentences of the documentation being analysed. However, I then

use the task phrases to identify relations between sentence instances, by connecting sentences that have task phrase words in common.

2.2.2.3 The Witt database

The Witt database is a product of a research project published in 2018 by Treude et al. which uses data from Stack Overflow and Wikipedia to develop an automated approach for the categorisation of software technology concepts [5]. An extension of this research is the ability to generate dynamic lists of all technologies of a given type. A list generated using this automated approach is stored in the Witt database and is used to match technology concepts that appear in the documentation.

3

Methods

This section goes over the methods used for each part of the research, from data collection and annotation and how the data is used to implement the automatic software knowledge extraction, to the creation and evaluation of Software Documentation Supporter (SDS), the system that was built to showcase and evaluate the results.

3.1 Research method

This research is a design science research and follows the design science research methodology (DSRM) for information systems. The methodology was formulated and presented by Peffers et al. [15] and consists of the following six steps:

1. Identify problem and motivate
2. Define objectives of a solution
3. Design and development
4. Demonstration
5. Evaluation
6. Communication

The problem was identified and motivated through existing research and was demonstrated in chapter 1, introduction and chapter 2, background. The objectives of a solution were then defined based on the problem identified and are demonstrated in the problem statement, chapter 1.4. In the design and demonstration step, the software knowledge categories and its structure, as well as the SDS were developed as a solution to the identified problem. In the demonstration step, data from a chosen case documentation was used to populate the software knowledge categories and the results demonstrated in the SDS. An evaluation was conducted on the SDS through usability tests and finally, the results are communicated through this report.

3.2 Data collection

In the beginning of this research, I got a hold of a dataset that was supposed to contain 100 software documentation files. However, it turned out that only 7 of those 100 files were actual software documentations. The files were extracted from Stack Overflow using a machine learning algorithm, unfortunately the accuracy of the algorithm was not as high as expected.

In order to create ground truth for the classification algorithm, all data needed to be annotated. As the data was already minimal, it was decided to only look at sentence level classifications. Annotating sentences into categories of knowledge however is an incredibly time-consuming task and resulted in the inevitable decision of only working with one documentation.

The software documentation of an e-commerce system called Snowflake, written by Escoriza [6] was chosen based on the quality of the content, structure and text. The documentation follows the IEEE standards for software documentation [16, 17, 18] quite thoroughly and contains information about requirement analysis, quality assurance, software architecture and other implementation details. The documentation was quite extensive and provided us with about 830 sentences to analyse.

3.3 Software knowledge categories and data annotation

Qualitative content analysis was used to annotate the sentences into software knowledge categories. As these knowledge categories were not known beforehand, an iterative bottom-up approach was used to obtain concrete definitions of the software knowledge categories that can be recognised in software documentation. In the beginning, the categories were inspired by existing models by both Tao et al. [11] and Soliman [19], but at this point it was unclear how well those categories would generalise to the subject of this research. A part of the dataset was classified using the initial categories. As the annotation process proceeded, some categories were deemed unnecessary while others emerged and the definition of the categories became more explicit.

When the knowledge categories and their definitions had become clear, a random sample of the dataset was then annotated by an independent party using these definitions. All deviations between those annotations and the ones made by the researcher were analysed and used to make category definitions more concrete.

3.4 Classifying sentences into software knowledge categories

Natural language data is very diverse and contains a lot of possible features to use for training a machine learning algorithm. When working with textual data, it is therefore important to use pre-processing techniques to deal with this diversity and make it easier for the machine learning algorithms to identify the useful features of the data. Various pre-processing methods were applied to the dataset, in order to remove inferior features.

Firstly, the diversity of the data was reduced by making all data lower-case and

removing digits and punctuation marks. The Natural Language Toolkit (NLTK) [26] platform was used for removing stop words, lemmatisation, and stemming. Stop words are words that occur frequently in text but do not bear any specific meaning. These words are often removed from textual data as they are not good candidates for features and might distract the machine learning algorithms. Lemmatisation removes the affix of the words and stemming then reduces words to their word stem [26].

This is a multi-class classification task, where every sentence should be classified into one of 25 categories. All Scikit-learn classifiers can handle multi-class tasks, so a collection of different classifiers were trained, tested and compared on the data set. As the software knowledge category structure has hierarchical properties, two classification approaches were also compared, flat classification and hierarchical classification. In the flat approach all data points are classified straight into the leaf nodes of the hierarchy, however in the hierarchical approach, a separate classifier is trained for each level of the hierarchy. For each trained classifier, the following Scikit-learn classifiers were compared.

Table 3.1: Scikit-learn classification algorithms evaluated for each trained classifier

Scikit-learn classification algorithm
<code>sklearn.naive_bayes.BernoulliNB</code>
<code>sklearn.ensemble.ExtraTreesClassifier</code>
<code>sklearn.neighbors.KNeighborsClassifier</code>
<code>sklearn.svm.LinearSVC</code>
<code>sklearn.linear_model.LogisticRegression</code>
<code>sklearn.neighbors.NearestCentroid</code>
<code>sklearn.ensemble.RandomForestClassifier</code>
<code>sklearn.linear_model.SGDClassifier</code>

When comparing the different classification algorithms, various metrics were measured and monitored.

Cross-validation accuracy. Cross-validation is a method used for splitting data into training and testing data. The dataset is split into k smaller sets. The model is then trained on $k-1$ sets and tested on the 1 remaining set. This is repeated k times, so each set is used as a testing set once. Using cross-validation is often considered a good approach to detect and minimise overfitting, however in this case, with a very little dataset, only using cross-validation accuracy was not enough, and therefore other metrics were monitored as well.

The dataset was then split into a training, testing and validation set. First the whole dataset was split into three parts, a 67.5% training set, 25% testing set and a 7.5% validation set. The model was first configured to using the training and testing set. When the optimised model had been found it was then validated using the held-out validation set. For this approach, the following metrics were measured.

Precision. Precision measures the model’s ability to identify only the relevant data points, and is defined by the following formula.

$$precision = \frac{TruePositive}{TruePositive + FalsePositive} \quad (3.1)$$

Recall. Recall measures the model’s ability to identify all the relevant data points, and is defined by the following formula.

$$recall = \frac{TruePositive}{TruePositive + FalseNegative} \quad (3.2)$$

F1 score. The F1 score is a measure that takes into account both the precision and recall of a model, and is defined by the following formula.

$$F1 = 2 * \frac{precision * recall}{precision + recall} \quad (3.3)$$

Overfitting. Overfitting is a measurement for the generalisability of a model, if a model is overfitting the training data, it is unlikely to generalise well to other data. Overfitting can be identified by comparing training and testing results. Training accuracy that is much higher than the testing accuracy is a good indication of overfitting in the model, since it is performing much better on data it has seen than on data it has not seen. The following formula was therefore used to measure the overfitting of the models.

$$Overfitting = TrainingRecall - TestingRecall \quad (3.4)$$

3.4.1 Identifying relations between sentences

Two different methods were used to identify relations between sentences; clustering and comparing words of task phrases extracted from sentences.

3.4.1.1 Clustering

A feature is a collection of related functional requirements and thus a functional requirement always corresponds to some feature of the system. Use cases are then used to demonstrate how the system will fulfill a certain functional requirement. For this reason, it is possible to assume that each sentence in functional requirement or use case category, relates to one of the system’s feature.

The fact that the sentences will always relate to one of the features of the system, makes clustering algorithms a potential solution to identify those relations.

A model was created using a Scikit-learn’s TF-IDF vectoriser [27], which reduces the textual data to the data of which words appear and how often. This technique

however results in a very high dimensional representation of the text. Principal component analysis (PCA), a technique used to reduce the dimensionality of data while still retaining their correlation, was therefore used to reduce the dimension of the data to a 2-dimensional space.

A clustering algorithm was then used to identify correlations between sentences. The clustering algorithm is unsupervised with the exception of a manual input for the variable k . The variable represents the number of clusters you wish the algorithm to identify and was therefore set to 3, that is, the number of features in the Snowflake system, as stated in the documentation.

In order to evaluate the accuracy of the results, all sentences in the feature, functional requirements and use case categories were manually classified based on which feature they corresponded to. The manual and clustering results were then compared.

3.4.1.2 Extracting and connecting sentences' task phrases

In a paper from 2015 by Treude et al. [30] a system that extracts task phrases from software documentation was developed and presented. A task phrase is a verb associated with a direct object and/or a prepositional phrase. The case documentation, Snowflake, was run through this system, and the task phrases extracted for each of its sentences.

These extracted task phrases were then used to identify relations between sentences, by identifying sentences that had task phrase words in common.

3.4.2 Identifying technology concepts

In order to identify technology concepts data from the Witt database, created by Treude et al. [5] was used. The Witt database is a static database that contains a list of software technology concepts identified on Stack Overflow. This database was used to find both architecture patterns and programming languages mentioned in the documentation that matched concepts belonging to those two categories in the database.

3.5 User questions and responses

The goal of this research was to use the extracted software knowledge to answer common developer questions. Collecting the questions for the system was an iterative process. Firstly, the questions were mostly inspired by information and questions from other research [8, 24, 25]. This initial set of questions was used to understand what knowledge was important to extract from the documentation. Later on, the list was refined to also highlight the capabilities of the SDS.

To formulate the responses to the questions, first I had to determine the appropriate information needed to properly answer each question. There was no formal theory to rely on in this case and instead this process was iterative, where I determined what knowledge I believed was relevant to the question, got feedback from others and made improvements. The presentation of the responses in the SDS was then inspired by Tao et al. [11]. There, graphs were used to present the responses to similar user questions, with good success.

3.6 Creating the Software Documentation Supporter

The Software Documentation Supported (SDS) is an important evaluation tool. It was created to showcase the results, experiment with their presentation and to evaluate their usefulness. The SDS is implemented using the MVC architecture and therefore consists of a model component, view component and a controller component.

The view component is written in JavaScript, HTML, and CSS. It contains the user interface and simple logic to display the results. Through the user interface, the user can view the documentation and choose from a list of questions to ask about it. The relevant data is then received from the model component and is displayed to the user in the form of either simple text or graphs, which are drawn with the help of the DagreD3 library.

The controller component is written in Python and Javascript. It contains most of the business logic and is used to query the data stored in the model component. Given a question chosen by the user, the appropriate data is requested from the model component. The data is then manipulated and sent back to the view component.

The model component contains an ontology that stores the classification and clustering results, the Witt database and the Task Phrase Extractor results. The ontology was created and populated using the Protégé¹ ontology editor and then hosted on an Apache Jena Fuseki server, which serves as an endpoint for querying the ontology data. The Witt database is stored in a database file and queried with the SQLite3 Python library. The extracted task phrases are then stored in a simple text file and Python is used to retrieve and manipulate the data.

Figure 3.1 shows the architecture of the system. Table 3.2 then lists the libraries that were used in the implementation of the SDS and their purpose.

¹<https://protege.stanford.edu>

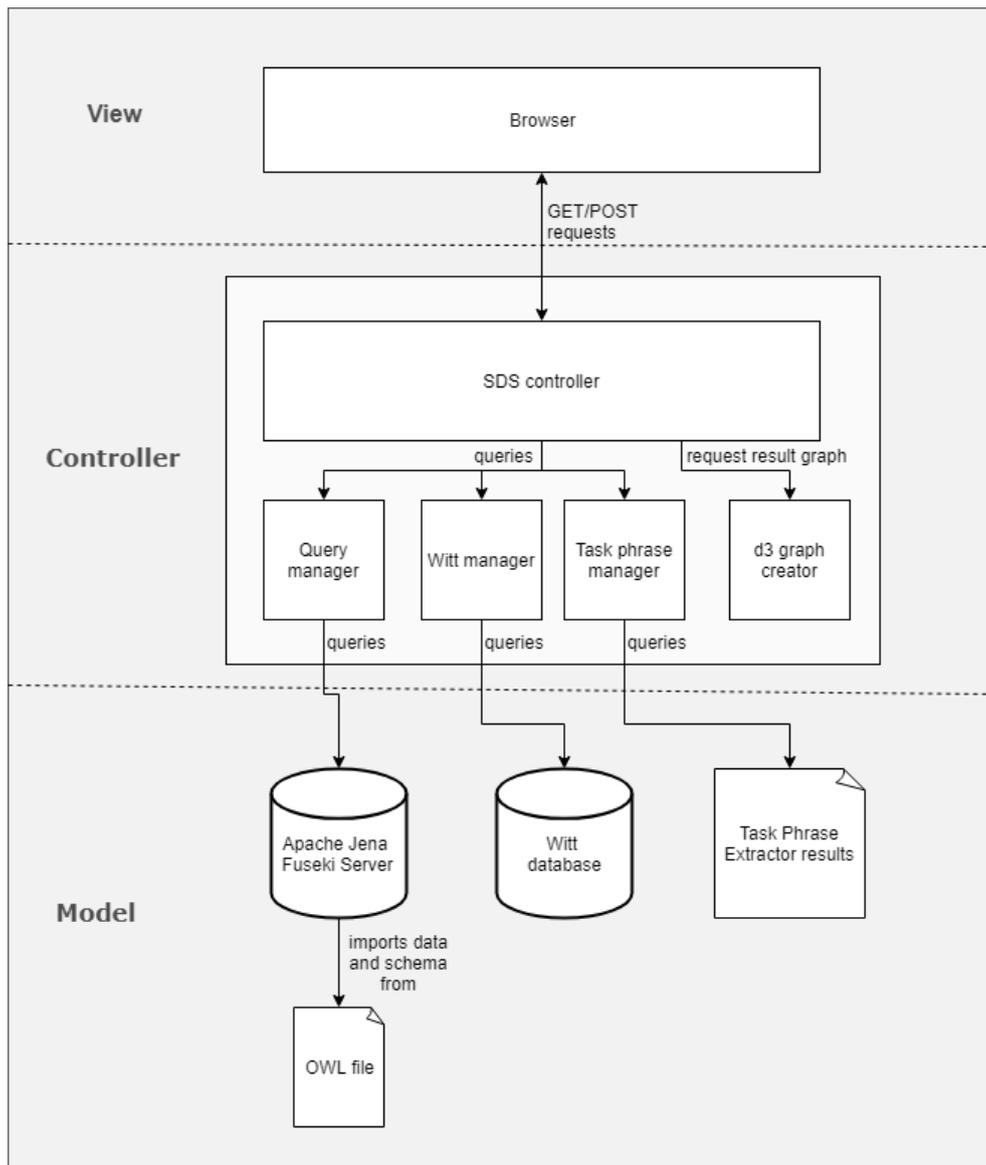


Figure 3.1: The architecture of the SDS

Table 3.2: Libraries used in the system implementation

Library	Purpose
Flask	A python web framework.
SPARQLWrapper	A Python library used to query the ontology.
SQLite3	A Python library used to query the Witt database.
JSON	Used to translate Python dictionaries and lists to a JSON format.
Bootstrap	A HTML, CSS and JavaScript library used to develop responsive web systems.
jQuery	A JavaScript library used for event handling.
DagreD3	A JavaScript library used to draw data structures as graphs.

3.7 System evaluation

The evaluation of the SDS was twofold. Firstly, I measured how efficiently users could solve tasks using the system compared to only using the software documentation itself. Secondly, qualitative user experience data was collected through a questionnaire and open-ended questions.

3.7.1 Participants

The participants of the usability tests were persons with background in software engineering and some experience related to software architecture, requirement analysis or the documentation of software.

All participants were asked to complete a personal experience questionnaire, which asked about level and field of education, years of experience within software engineering and level of experience of web development, software architecture and requirement analysis.

Nine people agreed to participate in the system evaluation. Of those nine participants, there were four professors or assistant professors from four different universities in three different countries, three doctorate and one master student from Chalmers University, and one person currently working in industry. Those nine individuals had an average of 7.8 years of experience in software engineering. See appendix E for more information on the participants' background.

One participant was excluded from the quantitative results of the usability tests, as they had extensive prior knowledge of the Snowflake documentation, which caused bias in the quantitative measures. The participant's feedback in the user experience interview was however very valuable and was included in the qualitative results.

3.7.2 Usability test

The usability tests consisted of a tutorial video and eight tasks.

The tutorial video was divided into two parts. The former was in the form of a slideshow and introduced the software documentation that they were supposed to analyse with and without the SDS, and then went over the basics of this research. The second part of the video was a demonstration of the SDS with textual explanations. The functionalities of the system were showcased to give the participants an idea of how to use it to efficiently navigate the documentation text.

The user then got eight tasks to solve, four using the system and another four using only the software documentation. In order to make the tasks compara-

ble, the system and documentation tasks where of similar nature, so for each task to be solved using the system, there was another very similar task, to be solved with the documentation. The tasks were created to represent how software developers use documentation in practice. The tasks of the usability tests are listed appendix D.

During the usability tests, quantitative data was collected in the form of task times.

Task time

The task time is the time it took the users to solve the task, measured from when they start reading the question, until they indicate they are done answering the question.

3.7.3 User experience interview

When the participants had solved all eight tasks, they were asked to give feedback on the user experience of the system.

Firstly, they were asked to fill in a questionnaire based on the System Usability Scale (SUS) by Brooke [28]. The SUS is a popular tool for measuring user experience and was chosen as it is quick and easy to use. Bangor et al. also showed that it effectively measures the usability of a system, even on small datasets [29].

Following the SUS questionnaire, the participants were then asked to answer four open-ended questions. The questions were designed to encourage participants to think about different aspects of their experience with the system. Having the questions open-ended then allow for broader and more extensive feedback from the participants. These questions made it possible to collect information about what the participants liked about the system and what ideas they have for improving or extending the SUS.

The user experience interview can be found in appendix D.

3.7.4 Evaluation sessions

Each evaluation session was carried out by a moderator who was responsible for following the predefined session schedule, measuring the task times and providing users with help where needed. Each session was scheduled to take about 60-70 minutes and was divided into three parts:

Table 3.3: Evaluation session schedule

Step	Time	Description
1	10-20 minutes	Participant watches a video tutorial that should teach them the basics of the system. They then play around in the system for a few minutes to get a better feeling of how it works.
2	40 minutes	Participant solves 4 tasks using the SDS and 4 similar tasks using only the written documentation (about 5 minutes per task).
3	10 minutes	Participant gives feedback on user experience through a questionnaire and four open-ended questions.

All sessions followed this same schedule, and all participants received the same information and solved the same tasks. However, there was some variation between the setup of the sessions. Firstly, three sessions took place in Sweden, two took place in Iceland and four sessions took place through Skype. The sessions in Iceland and through Skype were carried out by me whereas the sessions that took place in Sweden were carried out by a PhD student from Chalmers University. Lastly, half of the participants were asked to first solve tasks using the system while the other half started by solving tasks using the documentation. The purpose of this was to even the learning advantage that participants have when solving the latter four tasks.

4

Design and Implementation

This section has four main parts. Section 4.1 shows the defined software knowledge categories and discusses their design and structure. Section 4.2 discusses the implementation of the methods used to extract the knowledge from the software documentation. Section 4.3 explains the questions that users are able to ask in the SDS. Finally, section 4.4 demonstrates how the SDS answers those questions.

4.1 Software knowledge categories and their structure

The knowledge presented in software documentation is incredibly broad and every documentation is different in some aspect. It is therefore challenging to define categories that cover all knowledge presented in documentation. In order to support a wide range of possible software documentation, and to make possible future extensions of the software knowledge category graph easier, all categories were abstracted into high-level categories, presented in table 4.1.

The knowledge presented in software documentation can be abstracted to seven high-level categories. First of all there are three categories for information that is unclear or not tightly related to the system under development. Those are “Document organisation”, which contains knowledge about the structure of the document itself, “Non-information”, which contains knowledge that is uninformative in this domain, and “Uncertain” which contains knowledge that the annotators were not able to place in any category. Then there are four categories that cover all knowledge related to the system under development. Those are “Domain”, which contains knowledge about the environment in which the system exists in, “Requirement Analysis” which contains all knowledge related requirements and needs for the system, “System”, which contains all information related to architecture and implementation of the system itself, and finally “Development process”, which contains information related to the system’s development workflow and quality assurance.

These categories are listed and defined in table 4.1.

Table 4.1: High-level categories of knowledge presented in software documentation

Category	Definition
Domain	Describes or discusses concepts that are a part of the domain or environment in which the system exists in.
Requirement analysis	Describes or discusses the system's requirements analysis. Expresses a need for the system in question or describes requirements and their purpose in general.
System	Describes or discusses the system under development.
Development Process	Describes or discusses the process followed when developing the system and managing its quality.
Document Organisation	Describes the organisation of the document, e.g. what the next sections or sub-sections will discuss, what a particular diagram is showing, etc.
Non-information	Text that is presumed uninformative in this context. The sentence or fragment of text does not provide any relative knowledge.
Uncertain	Text that does not fall into any of the defined sections, or in some cases text that is cross-categorical, making it difficult to determine only one category for it.

All sentences of software documentation can be placed in one of those seven categories. Some sentences also express a directive or a rationale and further the rationale sometimes expresses a benefit or a drawback. Sentences can therefore get at least one of those four labels, in addition to their categorisation. The labels are listed and defined in table 4.2.

Table 4.2: Labels in software documentation

Label	Definition
Rationale	A justification of decisions made.
Directive	Describes constraints that have been imposed on the system, e.g. imposed by requirements or other technology being used.
Benefit	A benefit expressed or highlighted through a rationale.
Drawback	A drawback expressed or highlighted through a rationale.

The categories, labels, and their relations are shown in figure 4.1.

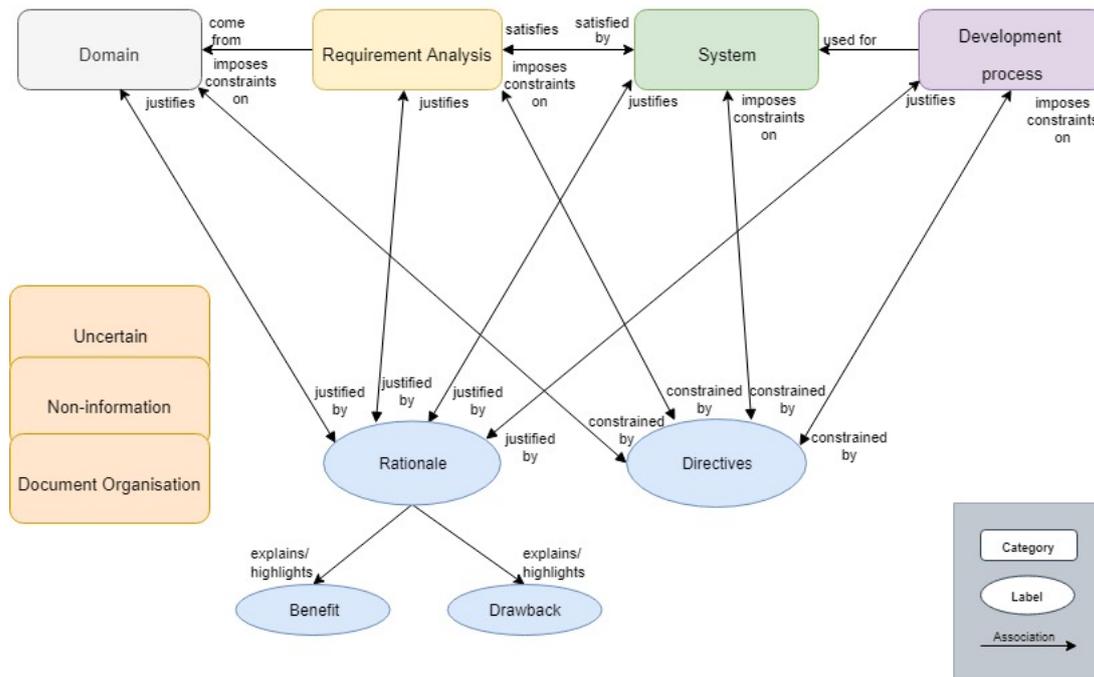


Figure 4.1: High-level view of the software knowledge category graph

A node in the software knowledge category graph represents knowledge about a system. Rounded box nodes denote categories while ellipses denote labels, and arrows denote relations between the nodes. There are two types of relations in the graph, association and inheritance. The high-level view of the graph however only contains association relations. Association expresses communication between two nodes and can be either one-directional or two-directional, based on the relationship between the two nodes. Inheritance however expresses generalisation, i.e. an IS-A relationship between two nodes.

The requirement analysis aims to identify the stakeholders' needs for the system under development, so all requirements originate from the domain. The domain and requirement analysis are therefore related with a one-directional “come from” association. The requirements of the requirement analysis are then satisfied by attributes of the system, and in the same way the system satisfies the requirements, and are therefore associated with a bi-directional “satisfies” and “satisfied by” association. The development process is used to ensure consistency in development practices and to manage the quality of the system, thus the development process is associated with the system with a one-directional “used for” association. The labels rationale and directives are associated with all the aforementioned categories. A directive has a bi-directional “imposes constraint on” and “constrained by” association with the categories and a rationale has a bi-directional “justifies” and “justified by” association with the categories. Benefits and drawbacks are then associated with rationale since in many cases, a rationale is explaining or highlighting a benefit/drawback of something, thus the benefits/drawbacks have a one-directional “explains/highlights” association with rationale. Finally, the categories uncertain, non-information, and document organisation all

contain information unrelated to the system under development and are therefore not associated with any of the categories.

Each of the abstracted categories shown in figure 4.1, divides into more detailed, lower-level categories. The categories of each section are listed and defined in table 4.3.

Table 4.3: Categories of knowledge presented in software documentation

Main-category	Sub-category	Definition
Domain	Stakeholder	Discusses particular stakeholders of the system and how the system will be useful for them.
Requirement	Functional requirements & behaviour	Describes what the system does or does not do in terms of functionality. The system under development is not mentioned in any detail. There is no talk of UI attributes like buttons, pages, etc, just a description of the functionalities that the system has to be capable of.
	Non-functional requirements & behaviour	Describes or discusses the non-functional requirements/quality attributes of the system.
	General	Describes requirements in general. The text does not state any specific requirements, but discusses something more general, related to requirements.
	User Story	Explains functional or non-functional requirements. Approximate format: “As a role, I want goal, so that benefit (priority).” Identified using rule-based learning on already identified functional/non-functional requirements.
	Feature	Describes a high-level objective of the system. Can usually be broken into several functional requirements. Example: “Purchase a product”
	Use Case	Describes how the system fulfills certain requirements. Here the system or some user-system interaction is mentioned in more details. Example: “The user is asked to fill a form with shipping information. He then presses the submit button to send the request”

System	Structure	Describes both the system's architectural structure, i.e. patterns, layers, classes, etc., and implementation structure, i.e. technology solutions/packages being used, and written source code.
	Structure - Architecture	Describes architectural attributes of the system, e.g. patterns, layers, classes and components.
	Structure - Implementation	Describes the implementation of the system structure.
	Structure - Implementation - Technology Solution	Describes out-of-the-box technology that is being used, e.g. programming languages, APIs, programming/development packages and frameworks
	Structure - Implementation - Source Code	Describes the source code implementation in some detail, e.g. methods and how they are written, executed or used.
	Behaviour	Describes the control-flow and communication between components/layers, etc.
	Behaviour - Architecture	Describes the architecture of the behavioural attributes of the system.
	Behaviour - Implementation	Describes how the system behaviour, i.e. control-flow and communication, is implemented.
	Behaviour - Implementation - Tech Solution	Describes out-of-the-box technology used to implement the control-flow or communication in the system. Example: A text describing how AJAX is used to send requests.
	Behaviour - Implementation - Source Code	Describes the source code written to implement the system's control-flow or communication.
	Behaviour - Implementation - General	Describes the implementation of control-flow or communication without going into details such as what technology is being used and how it is implemented in the source code.
	Data	Describes the data that is used or produced by the system.
	Data - Architecture	Describes the conceptual data model or architecture of the data used/produced by the system.
Data - Implementation	Describes the actual implementation of the data model in the system.	

4. Design and Implementation

	UI design	Describes the user interface in some detail. Sentences are e.g. likely to contain words like: button, page, display, structured, screen, navigation, scroll, design, HTML
	UI design - Architecture	Describes the user interface in terms of visualisation. Example: Where on the page is the submit button and what is its color.
	UI design - Implementation	Describes the implementation of the functional aspects of the user interface. Example: when is the page reloaded, when is input data submitted, etc.
Development Process	Development practice	Describes the workflow and general practice of developers, including how they do revision control, how they divide their work, etc.
	Quality Assurance (QA)	Describes or discusses how the quality of the system is ensured.
	QA - Issue	Describes identified faults, errors, bugs or some necessary improvements that need to be made.
	QA - Risk	Describes identified risks or pitfalls in the system's architecture or implementation.
	QA - Testing	Describes how the system in question will be tested, everything from usability tests to measuring performance of some components.

The categories and their structure are inspired by a research by Tao et al. [11] but have some differences. The requirement analysis segment of the category structure is similar to the one presented by Tao et al. The main difference is that test cases and stakeholders have been moved from this segment.

In order to make the system segment of the category structure support a wide range of software documentation, the 4+1 architecture used as inspiration. The 4+1 architecture model by Kruchten is a way of describing a system's architecture, using five different views [13]. Using this model as inspiration, the system segment is designed to cover all aspects of knowledge about the system under development.

Finally, the development process segment covers both development practice and quality assurance of the system. The development process has not been a part of the ontologies or knowledge categories reviewed for this research, however, this knowledge is not a concrete part of the system under development and is neither tightly linked to the requirement analysis and was therefore added

as a new main-category. The test cases that were part of the requirement analysis segment in Tao et al. were therefore moved to its quality assurance sub-category.

Figures 4.2 to 4.6 zoom in on different parts of the structure.

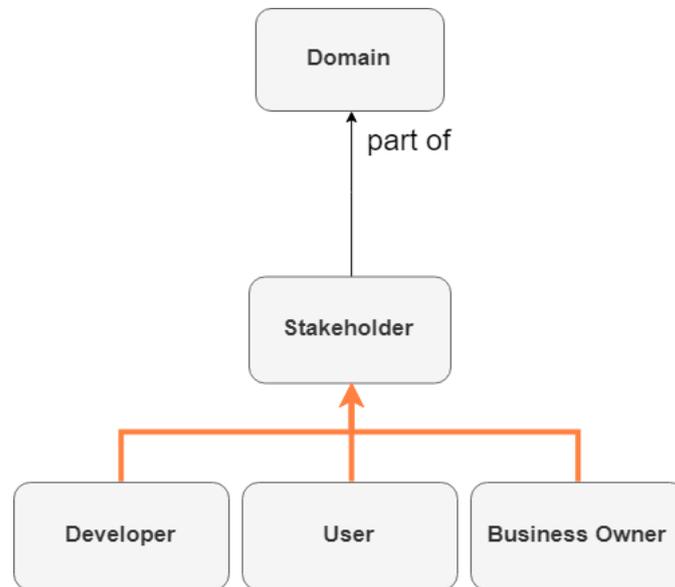


Figure 4.2: A view of the domain category and its subcategories

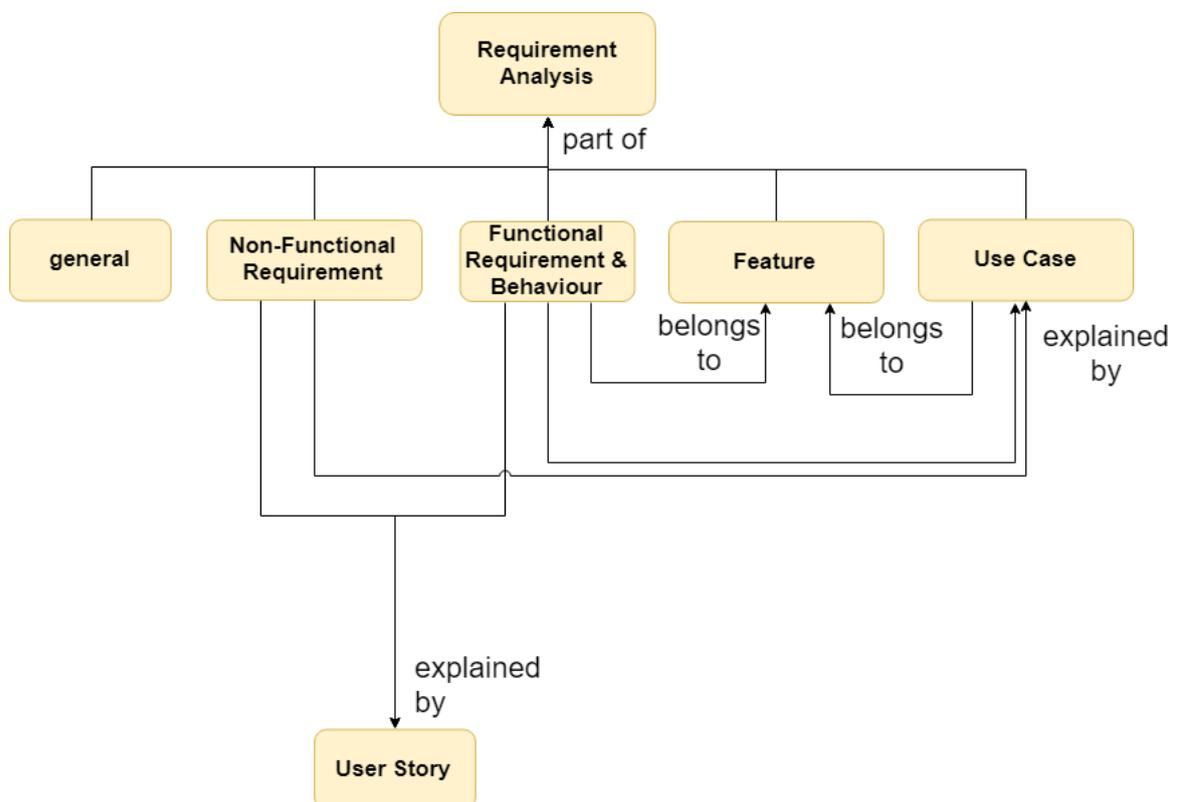


Figure 4.3: A view of the requirement analysis category and its subcategories

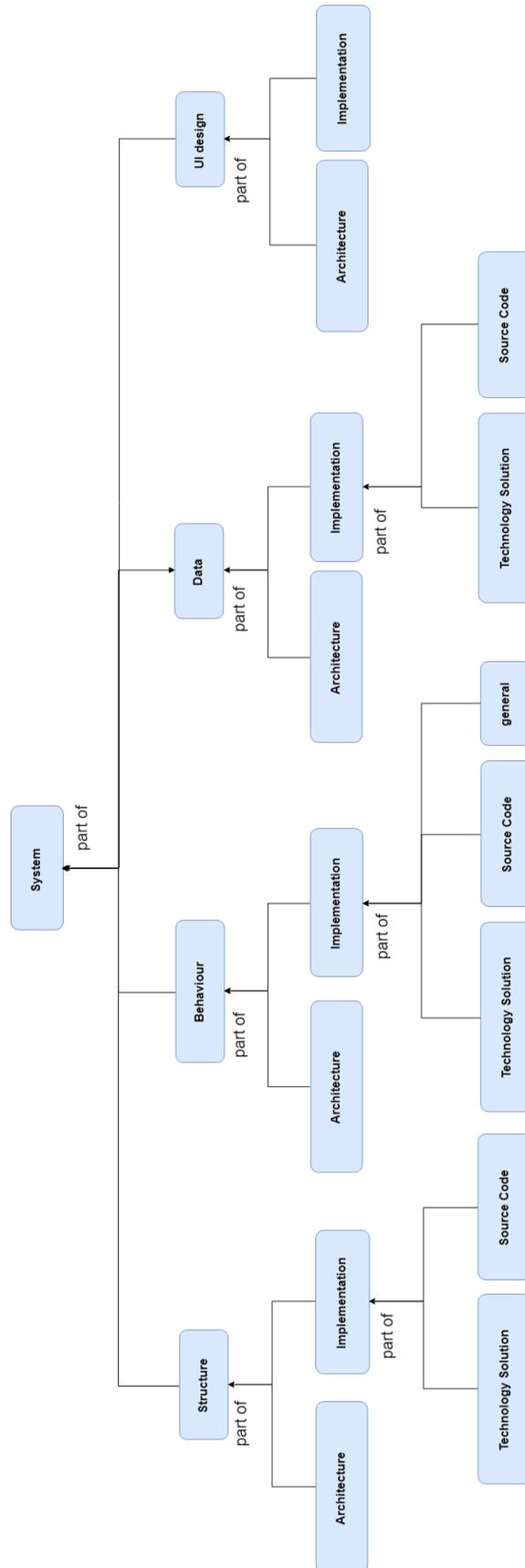


Figure 4.4: A view of the system category and its subcategories

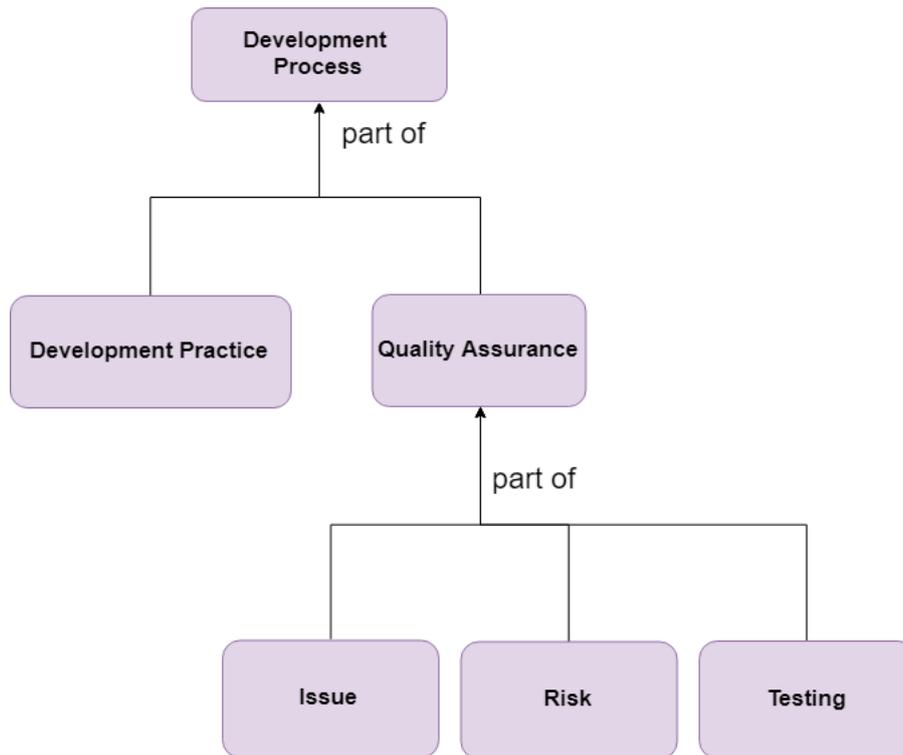


Figure 4.5: A view of the development process category and its subcategories

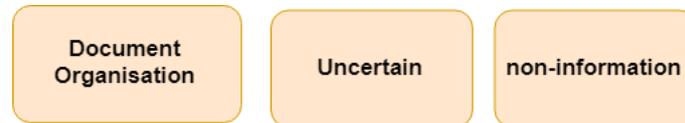


Figure 4.6: A closer view of other categories

4.2 Automatically extracting software knowledge

4.2.1 Category Classification

All sentences from the Snowflake documentation were classified with a supervised classification algorithm. Two different approaches, flat classification and hierarchical classification, were compared, and for both approaches, the performance of several different Scikit-learn classification algorithms were compared. Before applying the classification algorithm, all sentences were put to lower case, all stop words and digits removed, and all words lemmatised and stemmed, using the NLTK package.

4.2.2 Identifying sentence relations

Two different methods are used to identify sentence relations in the documentation. When every sentence of a certain category is related to exactly one instance in another category, clustering algorithms can be applied to automatically connect those sentence instances. This is exactly the case of functional requirements and

use cases, they are always related to exactly one feature instance. However, in most cases this does not hold and in those cases clustering algorithms do not perform well, as the number of clusters is unknown. In those cases, sentence relations are identified by using a Task Phrase Extractor by Treude et al. [30] to find commonalities between the sentences' task phrases.

The Task Phrase Extractor, extracts task phrases from sentences. A task phrase consists of a verb, an object and/or a prepositional phrase. Figure 4.7 shows an example of the task phrases extracted from the Snowflake documentation. Each task phrase is surrounded by curly brackets and each part of the task phrase is surrounded by square brackets and follows the schema [verb] [object] [prepositional phrase].

```
They can also provide some useful data for specific filters, like the maximum and minimum prices of a product list
for the price filter.
{{provide}} [useful data] [like minimum prices]{{provide}} [useful data] [like maximum]{{provide}} [useful data]
[for specific filters]}
Figure 5.18 shows the code that declares two of the filters used in this project with the corresponding keyword
used in the URL query string.
{{show}} [code] [ ]{{use}} [ ] [with corresponding keyword]{{use}} [ ] [in project]{{use}} [ ] [in URL query string]}
They are both included in the filter list that is directly bound to the request in the Figure 5.17.
{{include}} [ ] [in filter list]{{bind}} [filter list] [to request]}
The only piece missing to have a functional product search is the template rendering the search form, shown below
(Figure 5.19).
{{render}} [search form] [ ]}
The HTML input name and value are obtained from the fullTextSearch parameter defined before, hence connecting the
user filter interface with the SPHERE.IO filter request.
{{obtain}} [HTML input name] [from fullTextSearch parameter]{{obtain}} [value] [from fullTextSearch parameter]}
As can be observed, in a Scala template expressions to be evaluated with Scala are preceded by a @ sign.

This way classes can be imported and used directly in the template like is the case with the filter fullTextSearch.
{{use}} [way classes] [in template]{{import}} [way classes] [ ]}
It is also interesting to notice how URLs are generated, using the corresponding reverse routing method for the
controller action that handles searches.
{{use}} [corresponding reverse routing method] [for controller action]{{handle}} [searches] [for controller
action]{{handle}} [corresponding reverse routing method] [for controller action]{{generate}} [urls] [ ]}
Using the methods for sorting (Figure 5.20) and paging (Figure 5.21) is very simple, as the two pieces of code
above demonstrates.
{{use}} [methods] [for sorting]{{use}} [methods] [for paging]}
Sorting is achieved by specifying the correct sorting criterion, provided via a ProductSort enumeration that
contains all the possibilities currently offered by the SDK.
{{specify}} [correct sorting criterion] [ ]{{provide}} [ ] [via ProductSort enumeration]}
```

Figure 4.7: An example of the results produced by the Task Phrase Extractor

The words of the task phrases are then compared and all sentences that have task phrase words in common are assumed to relate to each other.

In this case, there is no ground truth to compare the results to. The evaluation of these results will be through usability testing, where the usability of the results is measured.

4.2.3 Retrieving technology concepts

Technology concepts were extracted from the documentation by identifying the concepts in the documentation that matched a technology concept stored in the Witt database by Treude et al. [5].

The Witt database contains a table *Categories* that stores tag-category relations. A tag is the name of a concept, e.g. “JavaScript” and the category is the

type of technology concept, e.g. “programming language”. The table has 40,400 rows, which contain 26,419 different tags that belong to 2,708 different technology concepts. An example from the Categories table is shown in figure 4.8.

ID	Tag	Category
1	javascript	language
2	javascript	typing
3	javascript	programming
4	javascript	programming-language
5	javascript	oo
6	java	language
7	java	programming-language
8	c#	sharp
9	c#	programming-language
10	php	programming-language
11	php	scripting
12	php	tool
13	php	scripting-language
14	android	os
15	jquery	software
16	jquery	library
17	python	programming-language
18	python	language

Figure 4.8: A screenshot from the Categories table in the Witt database

This table can be used to identify various different technology concepts. In this research it was used to identify architecture patterns and programming languages. The drawback of this approach is that some names of technology concepts are also common words in the English language. Therefore, the results can contain some false positives.

Four architecture patterns were identified in the documentation; workflow, state, composite and MVC. Of those four results, only one, MVC, was a true architecture pattern used in the system. Ten programming languages were identified; JavaScript, Java, Ruby, Swift, Scala, DOM, CoffeeScript, Scheme, Processing and Basic. However, DOM, Scheme, Processing and Basic were false positives. DOM is a programming interface and should therefore not be included as a programming language, but the other three were mixed up with the English words scheme, processing and basic.

4.3 The questions of the SDS

The user can choose from a list of 8 questions in the SDS. The questions are listed in table 4.4.

Table 4.4: The questions of the SDS

ID	Question
Q1	What functionalities exist in the system?
Q2	What functionalities does this feature provide?
Q3	How is this functionality implemented?
Q4	What was the development process related to this functionality?
Q5	What was the development process related to this non-functional requirement?
Q6	What architecture patterns are used in the system?
Q7	What programming languages are used in the system?
Q8	How are the architecture patterns in the system implemented?

Questions Q1 and Q2 focus on the requirement analysis of the documentation. These questions are useful as they give a quick overview and a high-level understanding of the system described in the documentation. They give the user information about the features, functional requirements and use cases of the system and how they relate. Q1 gives information about all the features of the system and what functional requirements and use cases belong to which features, whereas Q2 gives information about one certain feature, chosen by the user, and what functional requirements and use cases belong to that feature. When users ask this question, they are presented with a list of the systems features. This list was created manually and is independent of the classification results. To answer these questions, the classification algorithm was used to select the sentences of the functional requirements and use cases categories. The results from the Gaussian Mixture clustering were then used to determine which feature those functional requirements and use cases belonged to.

Questions Q3 and Q4 focus the functionalities of the system and retrieving information about how they are implemented, tested, etc. These questions were inspired by a previous study, as indicated in section 3.5, and are useful to get more

details about the system’s functionalities. When users ask any of these questions, they are presented with a list of the functional requirements of the system. Again this list of functional requirements was created manually and is independent of the classification results. Q3 targets the implementation of the functionalities and gives the user information about structural, behavioural and user interface aspects of the implementation of the functionality. This question also contains three sub-questions; “What is the behaviour of this functionality?”, “What is the structure of this functionality?” and “What is the UI design of this functionality?”. So if users were only interested in one of those implementation aspect of the functionality, they could choose the relevant sub-questions to get a more precise answer. Q4 then gives information about the development process related to the chosen functionality, that is, information related quality assurance such as testing and identified issues, or related to the development practice followed when implementing the functionality.

Question Q5 focuses on the non-functional requirements of the system. Those are for example requirements related to maintainability, performance and usability. Now the user is presented with a list of non-functional requirements to choose from, and again the list is manually created and independent of the classification results. The question is similar to Q4 and will give the user information about the development process of the functional requirements. For non-functional requirements, questions targeting the implementation are not included, since those requirements are in most cases not related to specific functionalities that exist in the system itself.

To answer questions Q3 to Q5, the Task Phrase Extractor, described in section 3.4.1.2 was used to extract task phrases from sentences of the documentation and its words then matched to find sentences that relate to the chosen functionality. The sentence classification was then used to filter the results based on the question asked by the user.

Finally, Q6 to Q8 focus on specific technology concepts mentioned in the documentation. Those questions are believed to be useful for architects and developers and are also examples of questions that can be very tedious to answer using only written documentation. Q6 and Q7 will give a list of the architecture patterns and programming languages mentioned in the documentation and Q8 will give information about how the architecture patterns of the system are implemented. To answer these questions, the Witt database, described in section 3.4.2, was used to identify the concepts. Additionally, to answer Q8, the Task Phrase Extractor and sentence classification was used to find implementation information related to the architecture patterns.

4.4 System demonstration

The SDS is a web system. Its structure is simple and the system only contains one web page, so all of the system’s functionalities can be carried out on this one page.

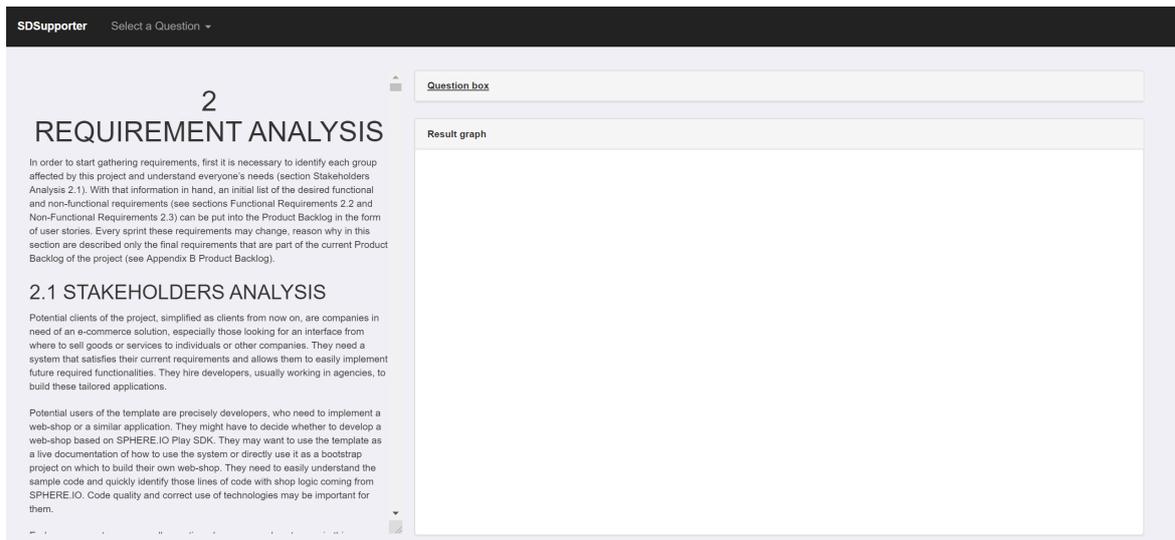


Figure 4.9: A screenshot showing the SDS when first opened

Figure 4.9 shows the website as it appears when first opened. On the left side, the documentation to be analysed is displayed. On the right side there are two boxes that are used to display the answers to questions. The horizontal ratio between the documentation section and result section can be adjusted. Users can therefore decide to make the documentation section width larger when they are reading the text, and likewise, they can make the result section width larger when they are reading information from the result boxes. The navigation bar then contains a drop-down menu, shown in figure 4.10 where users can choose a question to ask about the documentation.

As mentioned above, the results are displayed in the two boxes on the right side of the page. The upper box contains information about the question asked by the user and the categories and connection words used to formulate the result graph. The categories and connection words are displayed as buttons which can be used to filter the results displayed in the graph. A mock-up of the upper result box is shown in figure 4.11

Select a Question ▾

Features and functionalities
 What functionalities exist in the system?
 What functionalities does this feature provide?

Functional requirements
 How is this functionality implemented?
 --- What is the behaviour of this functionality?
 --- What is the structure of this functionality?
 --- What is the UI design of this functionality?
 What was the development process related to this functionality?

Non-functional requirements
 What was the development process related to this non-functional requirement?

Technology concepts
 What architecture patterns are used in the system?
 What programming languages are used in the system?
 How are the architecture patterns in the system implemented?

Figure 4.10: A drop-down menu containing the questions that the user can ask about the documentation

Question asked by the user

The feature/functional requirement/non-functional requirement chosen by the user

Categories shown in the graph: Words used to connect:

Category 1 Category 2 Category 3 Word 1 Word 2

Figure 4.11: A mock-up of the upper result box

The lower result box contains the results themselves. The results are the sentences that are considered relevant to the question asked by the user. The sentences are displayed as a graph to clearly show the feature or connection word that each sentence relates to. They are also displayed within a box that indicates its category. A mock-up of the result graph is showed in figure 4.12.

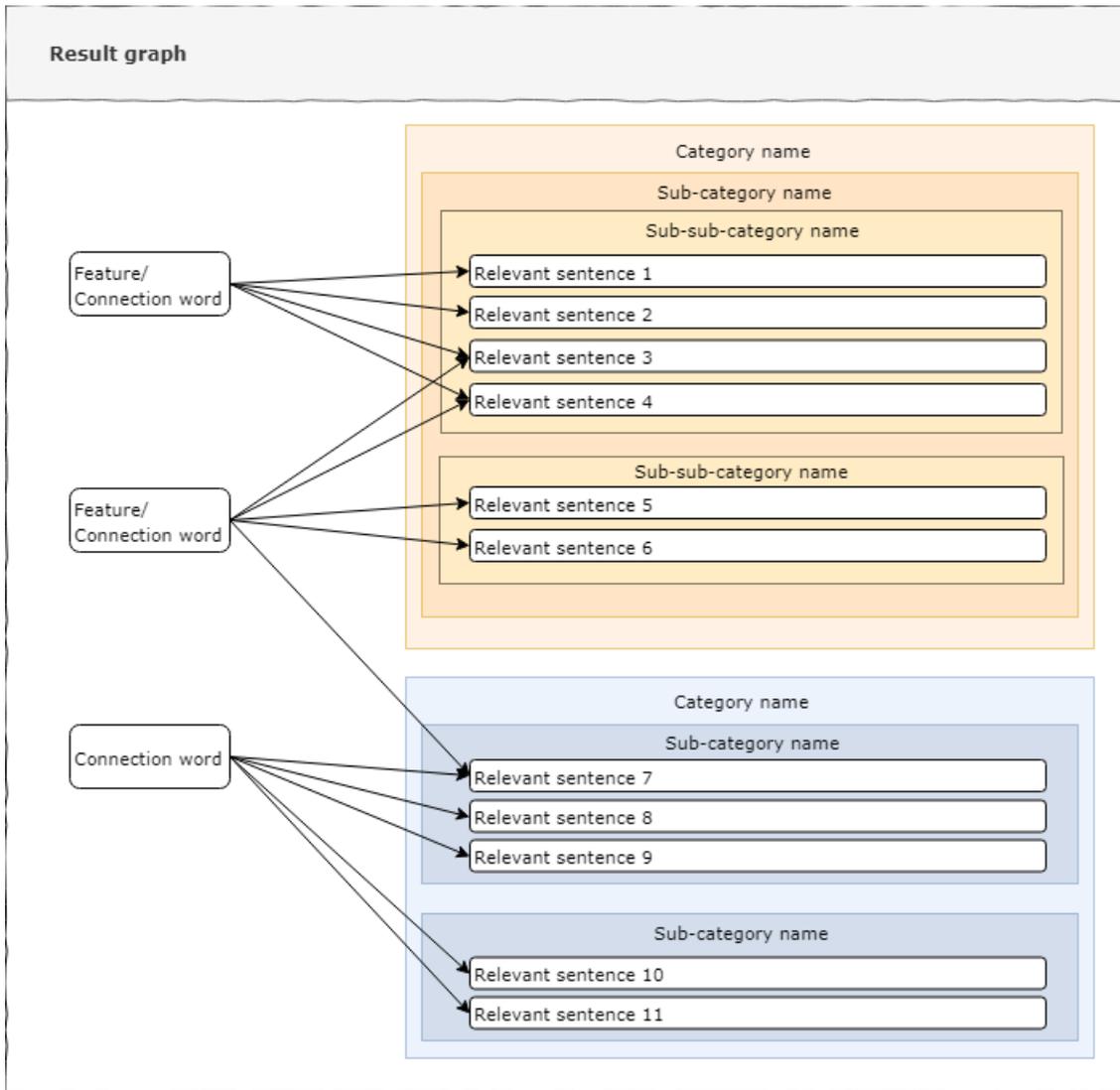


Figure 4.12: A mock-up of the lower result box

The results are always sentences or words extracted from the documentation text and all sentences that are a part of the results are highlighted in the documentation text. To ensure traceability, and to allow the user to get better context, clicking the resulting sentences or words will take the user to the place in the documentation from where the text was retrieved. If the result is a word and it appears multiple times in the text, the user can click it again to review the next place where the word appears.

The format of the results varies slightly depending on the question that is asked by the user. Sections 4.4.1 to 4.4.8 show the results to each question in more detail.

4.4.1 Q1: What functionalities exist in the system?

This question gives the user information about the functionalities of the system and to which feature they belong. This information should help the user to get a high-level understanding of how the system works.

The data needed to answer these questions are the functional requirements and the use cases, as those two categories contain all information about the functions that the user can carry out when using the system. For each functional requirement and use case, data about what feature they belonged to was also retrieved, to correctly connect the sentences to a feature of the system.

The results for this question are quite extensive as they include all functionalities mentioned in the documentation. However, the user is able to filter both the categories and the features, to make the results more focused. Figure 4.13 shows the results to Q1, after filtering to only see use cases belonging to the display product feature.

4.4.2 Q2: What functionalities does this feature provide?

This question is very similar to Q1, but now the user is asked to choose the feature of interest beforehand. When the user asks the question, a drop-down list of the possible features is therefore displayed (see figure 4.14).

The data needed to answer this question is again functional requirements and use cases, as they contain the information about the system's functionalities. But now the only information retrieved are the functional requirements and use cases that belong to the feature that was chosen by the user.

As the user has already chosen the feature of interest, this time the categories included in the graph is the only filter.

4.4.3 Q3: How is this functionality implemented?

Everything related to the implementation of a certain functionality is included in the result when the user asks this question. Now the user can get more detailed information about a certain functionality. When asking this question, the user

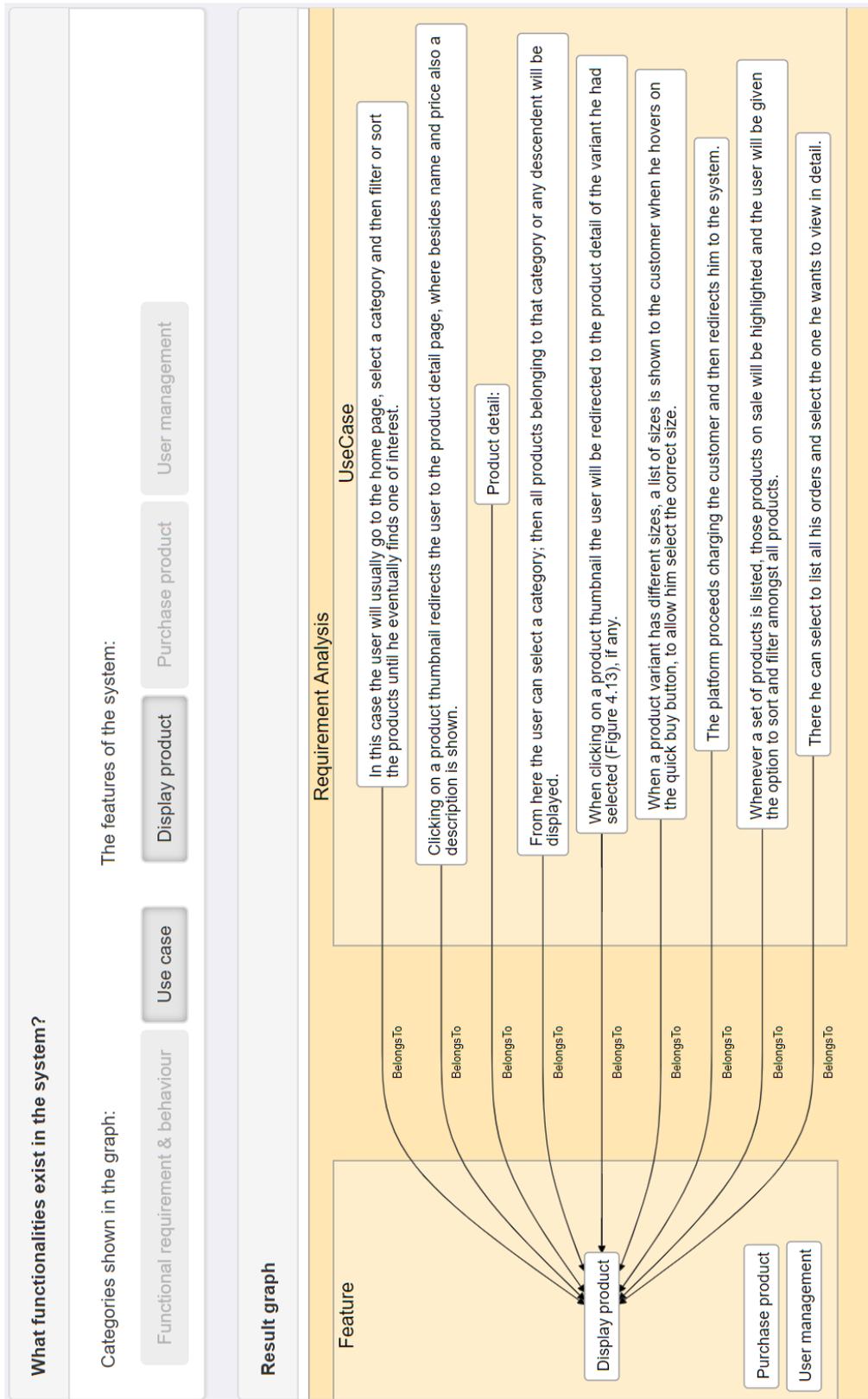


Figure 4.13: A screenshot showing results to Q1, filtered to show only use cases for the display product feature

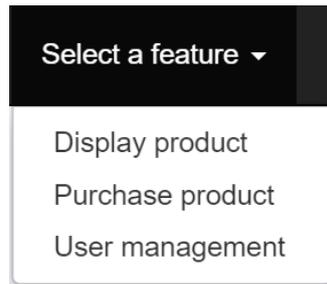


Figure 4.14: A screenshot showing the feature selector

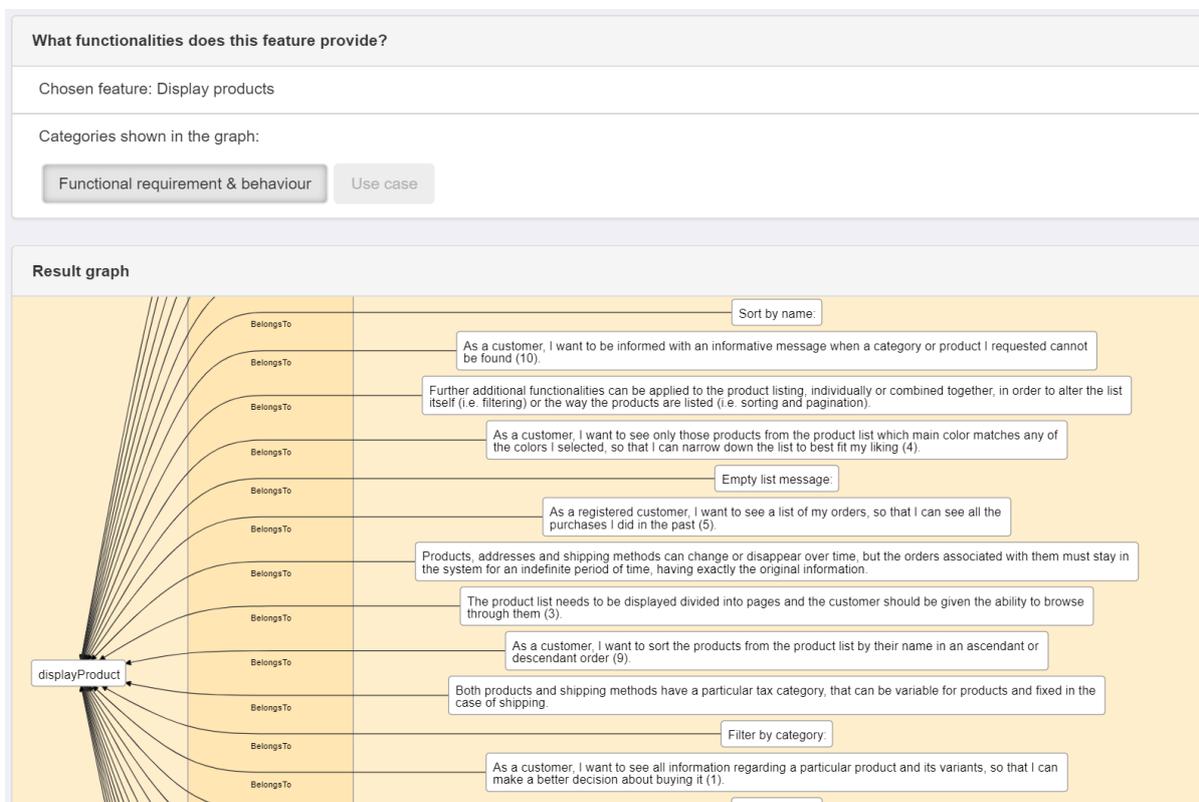


Figure 4.15: A screenshot showing a part of the results to Q2

therefore also has to choose the functional requirement of interest, from a drop-down menu (see figure 4.16) that is displayed when the question is chosen.

4. Design and Implementation

Select a user story ▾

Display products

As a customer, I want to list all products of the shop (1).

As a customer, I want to see only those products that belong to a particular category and any category descendant, so that I can narrow down the list to what fits best my needs (2).

As a customer, I want to see only those products from the product list which prices fall within a specific price range, so that I can narrow down the list to best fit my economic requirements (3).

As a customer, I want to see only those products from the product list which main color matches any of the colors I selected, so that I can narrow down the list to best fit my liking (4).

As a customer, I want to sort the products from the product list by their name in an ascendant or descendant order (9).

As a customer, I want to sort the products from the product list by their price in an ascendant or descendant order (8).

The product list needs to be displayed divided into pages and the customer should be given the ability to browse through them (3).

As a customer, I want to see all information regarding a particular product and its variants, so that I can make a better decision about buying it (1).

As a customer, I want to be informed of my location inside the category tree via a breadcrumb, so that it can help me to navigate and have a better understanding of the web-shop structure (6).

As a customer, I want to be informed with an informative message when a product list request has no results (5).

As a customer, I want to be informed with an informative message when a category or product I requested cannot be found (10).

Purchase products

As a customer, I want to add a particular product to the shopping cart, so that I can buy it with the next order (1).

As a customer, I want to change the number of units of a particular item in the shopping cart, so that I can buy a different quantity of the product with the next order (6).

As a customer, I want to remove a particular item from the shopping cart, so that I do not buy it with the next order (3).

As a customer, I want to place an order, so that I can actually buy the items in my shopping cart (2).

As a customer, I want to be able to pay online my orders, so that I can pay immediately the moment I buy them instead of using other possibly unpleasant billing options (4).

As a registered customer, I want to see a list of my orders, so that I can see all the purchases I did in the past (5).

As a customer, I want to be able to see my current shopping cart from any page via a so-called mini-cart, so that I can always be aware of its contents and pricing details (5).

User management

As an anonymous customer, I want to sign up a new customer account, so that I can place orders more easily and take advantage of many other benefits (4).

As an anonymous customer, I want to log in with an existing customer account, so that I take advantage of the benefits of being a registered customer (4).

As a registered customer, I want to logout from my customer account, so that nobody else can use it from the same machine (5).

As an anonymous customer, I want to be able to recover my password, so that I can log in with my account when I forget my current password (7).

As a registered customer, I want to update my personal data such as the email address used (6).

As a registered customer, I want to change my current password to another one of my choice (5).

As a registered customer, I want to add a postal address to my address book, so that I can select it as shipping or billing address when placing an order (5).

As a registered customer, I want to update the data of a particular postal address from my address book, so that it corresponds to my current situation (6).

As a registered customer, I want to remove a particular postal address from my address book, so that I cannot longer select it when placing an order (5).

Figure 4.16: A screenshot showing the functional requirement selector

To answer this question, all sentences belonging to the sub-categories of the System category are retrieved from the database. The Task Phrase Extractor is then used to find both the task phrase of the functional requirement chosen by the user and of all the retrieved sentences. The words of the task phrases are then compared and every sentence that has a task phrase word in common with the chosen functional requirement is considered relevant to answer this question.

Figure 4.17 shows the results to this question. Here the chosen functional requirement was “As a customer, I want to list all products of the shop.” The task phrase of the sentence is shown in bold in the upper result box and those are the words that are used to create connections to other sentences in the documentation.

In this case, the user is most likely interested to know more about the system's implementation when it comes to *listing products*, so here it is important to filter the results so that they are more focused on the results that the user is truly interested in. The word *shop* has therefore been disabled, as it is unlikely to give you results relevant to *listing products*, but the word *products* has also been disabled as in this domain it is very general and is likely to bring a lot of noise to the results.

Q3 then has three sub-questions. Those give very similar results, except they are more focused on certain parts of the implementation. Q3-1 therefore only gives the user information about the behavioural aspects of the functionality's implementation, Q3-2 only shows information about structural aspects of the functionality's implementation and Q3-3 only shows information about the implementation of the user interface. The results to questions Q3-1 to Q3-3 are shown in figures 4.18 to 4.20

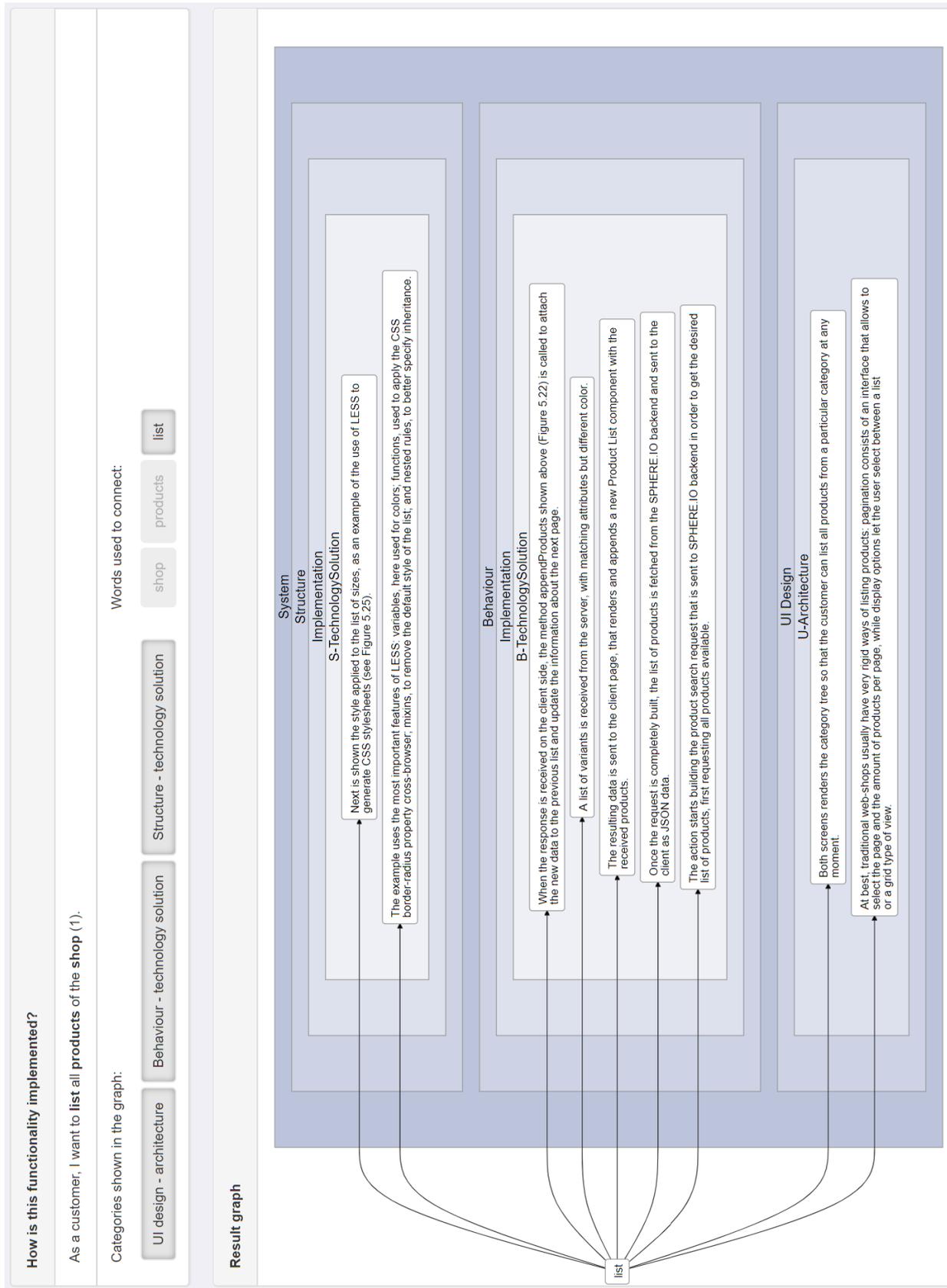


Figure 4.17: A screenshot showing results to Q3



Figure 4.18: A screenshot showing results to Q3-1

What is the structure of this functionality?

As a customer, I want to **list** all **products** of the **shop** (1).

Categories shown in the graph: **Structure - technology solution** **shop** **products** **list**

Result graph

System Structure

Implementation

S-TechnologySolution

It is also necessary to use the CLI in order to manipulate and query data in batches or for automated jobs, such as importing products into SPHERE.IO.

Next is shown the style applied to the list of sizes, as an example of the use of LESS to generate CSS stylesheets (see Figure 6.25).

The example uses the most important features of LESS: variables, here used for colors; functions, used to apply the CSS border-radius property cross-browser; mixins, to remove the default style of the list; and nested rules, to better specify inheritance.

Figure 4.19: A screenshot showing results to Q3-2

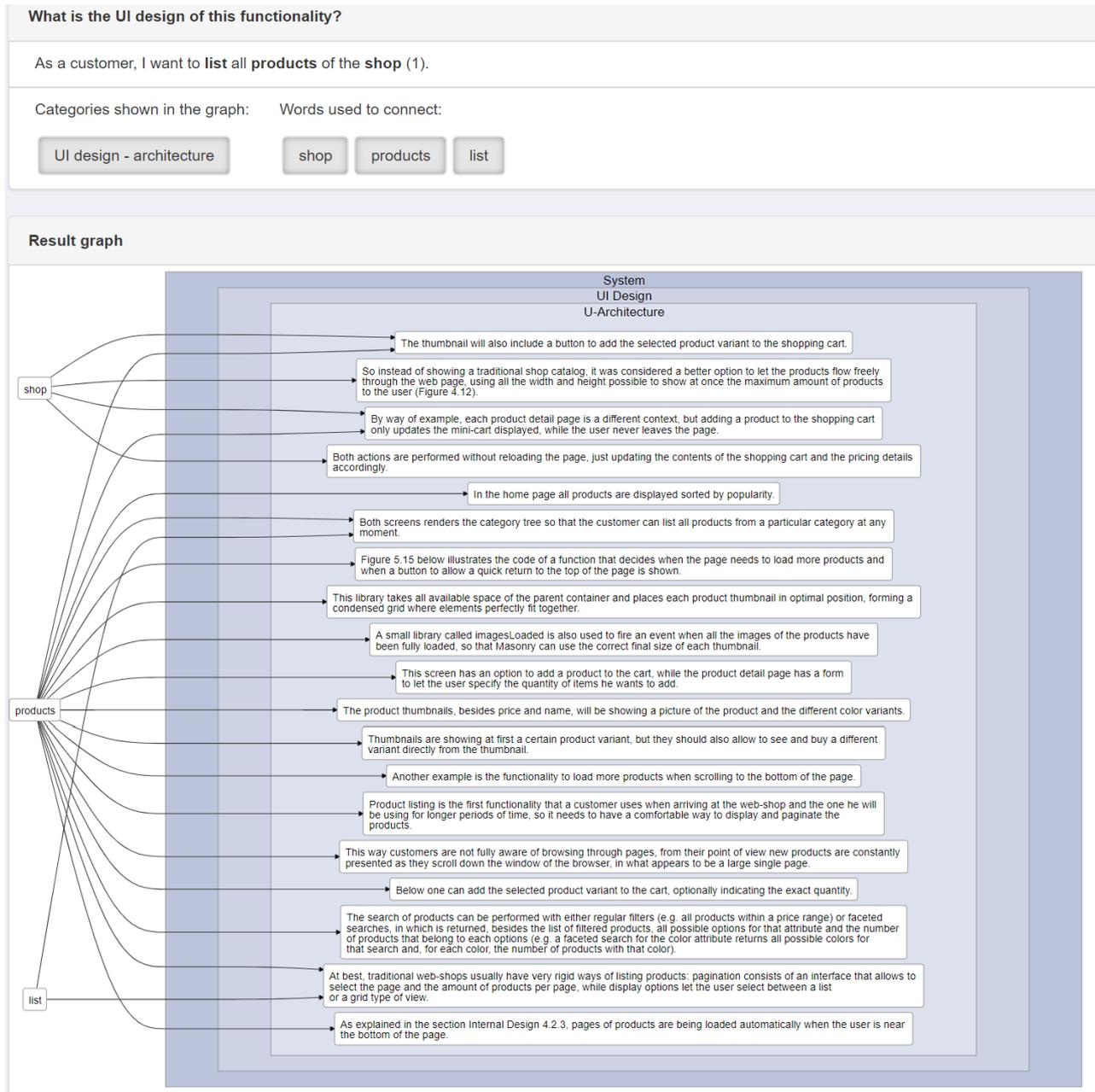


Figure 4.20: A screenshot showing results to Q3-3

4.4.4 Q4: What was the development process related to this functionality?

In this question, everything related to the development practice and quality assurance of the functionality is considered relevant. Through this question, the user can get information about for example the testing process or identified issues related to a certain functionality. As in Q3, the user also needs to choose the desired functional requirement.

To answer this question, all sentences belonging to the sub-categories of the “development process” category are retrieved from the database. The Task Phrase Extractor is then used to find the sentences that relate to the functional requirements chosen by the user.

An example of the results for this question can be seen in figure 4.21.

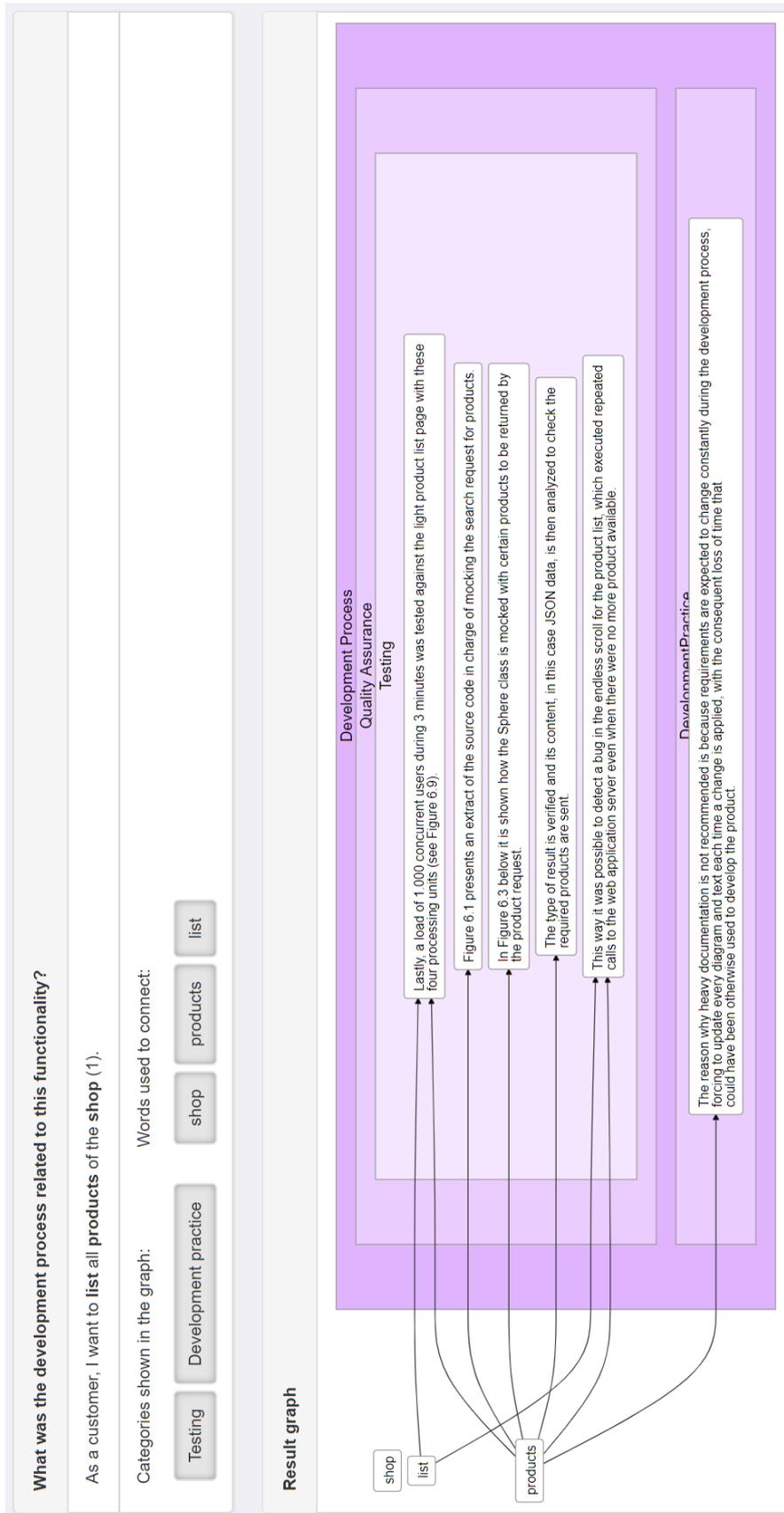


Figure 4.21: A screenshot showing results to Q4

4.4.5 Q5: What was the development process related to this non-functional requirement?

Non-functional requirements usually do not discuss actual functionalities that users can carry out in the system, but instead focus on its qualitative attributes, like for example performance and maintainability. Those requirements are therefore not something that is implemented in the system, but is tested and improved in various ways. Most information about non-functional requirements is therefore within the sub-categories of the “development process” category, and thus, this is the only question about non-functional requirements that is included in the system.

When asking this question, the user has to choose the desired non-functional requirement from a list that is displayed as a drop-down menu (see figure 4.22).

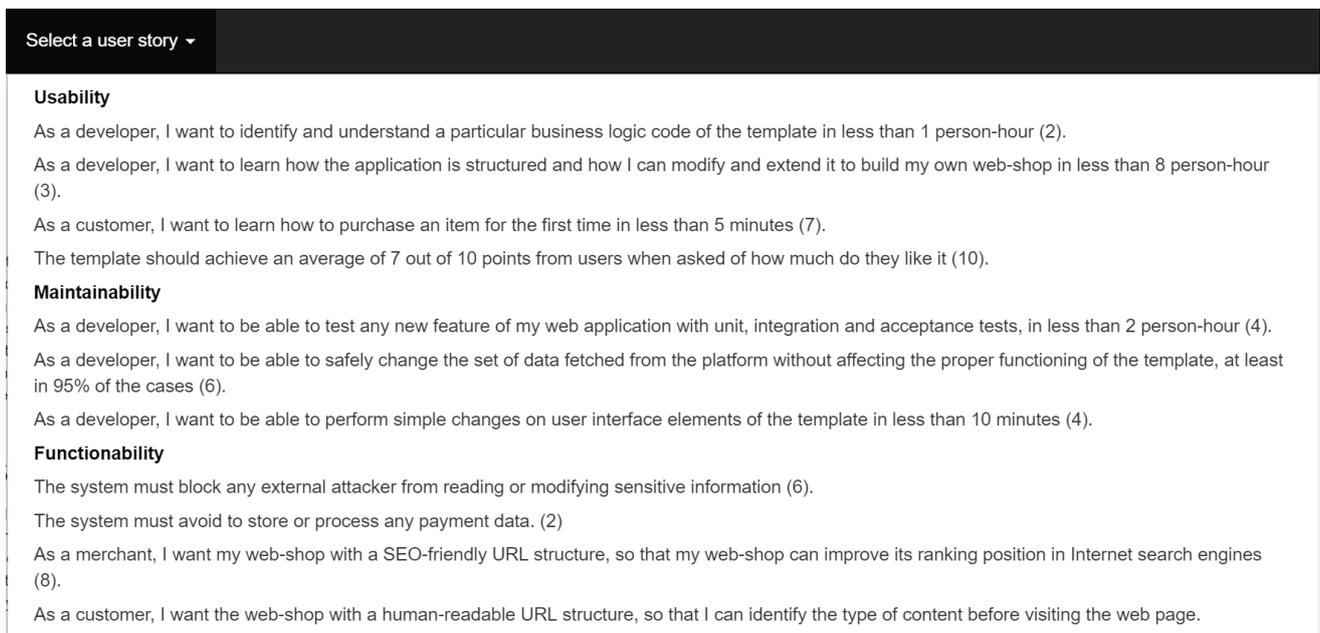


Figure 4.22: A screenshot showing the non-functional requirement selector

An example of the results for this question can be seen in figure 4.23. In this example, the chosen non-functional requirements discussed unit, integration and acceptance testing. However, the extracted task phrases contained quite many words that are not of interest if the user is solely interested in more information about those tests. Therefore it is important to disable and filter out the words that do not relate to the tests, and even to also disable to word *test*, as it is a quite general word that can bring noise to the results.

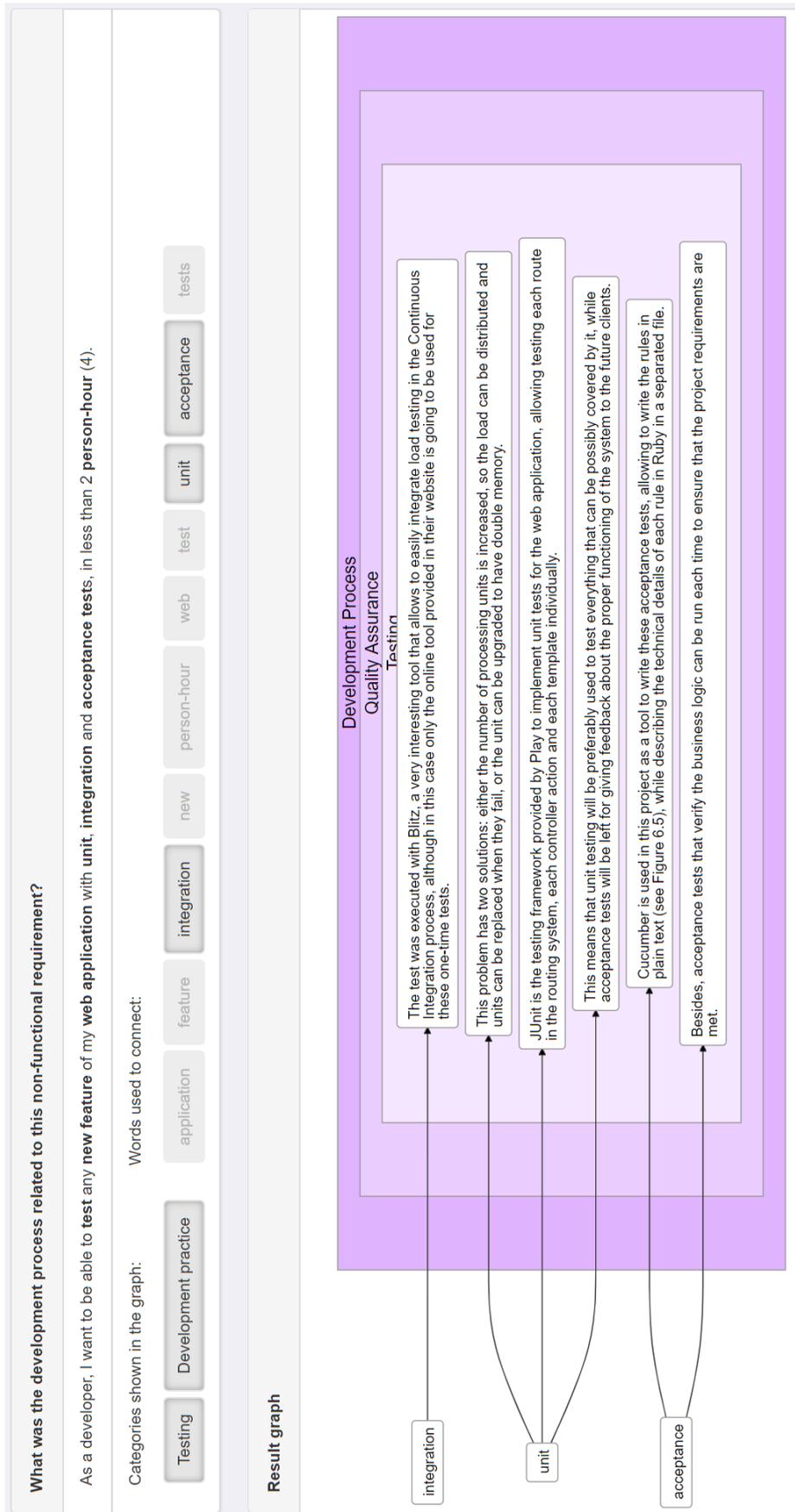


Figure 4.23: A screenshot showing results to Q5

4.4.6 Q6: What architecture patterns are used in the system?

The answer to this question is a simple list of all possible architecture patterns used in the system implementation.

To answer this question, a simple word match was implemented, using the Witt database as a source for architecture pattern names. Since some pattern names are also common English words, the word *possible* is used in the results to indicate that the system is not 100% sure of the context, and the user should quickly verify the results. To make this easier, the resulting words are clickable, and clicking them will take the user to the sentences that contain the words, so the user can quickly review the context of the sentence to confirm whether this is indeed an architecture pattern.

Figure 4.24 shows the possible architecture patterns identified in the documentation. After reviewing the results in the documentation text itself, it is clear that there is one architecture pattern used in the system and that is MVC.

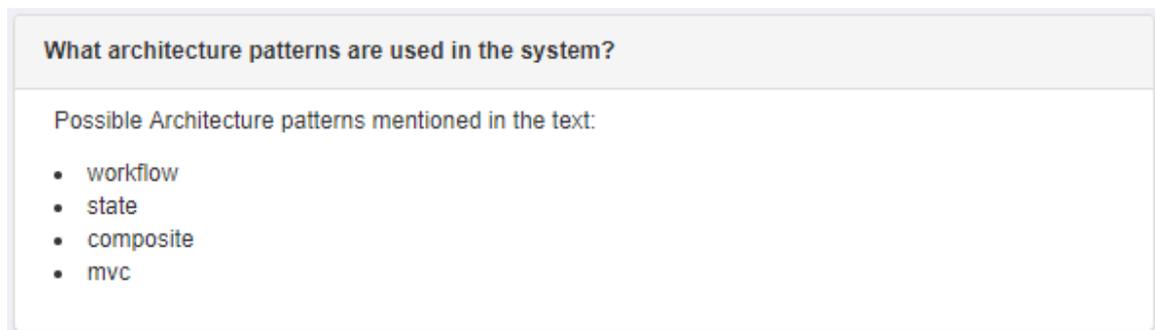


Figure 4.24: A screenshot showing results to Q6

4.4.7 Q7: What programming languages are used in the system?

This question is very similar to Q6, but now a list of possible programming languages used to implement the system is retrieved and displayed to the user.

Again, the user should verify the results by checking the context of each programming language. In this documentation, *scheme*, *processing* and *basic* are not programming languages. On top of that, there is also one false positive in the results as *DOM* is not a programming language. This error is the result of *DOM* being wrongly listed as a programming language in the Witt database.

4.4.8 Q8: How are the architecture patterns in the system implemented?

This question is similar to Q6 as it finds all the architecture patterns in the system, however it also finds information about implementation details of the possible architecture patterns.

Figure 4.26 shows the answer to this question. Due to a combination of low categorisation accuracy and the Task Phrase Extractor sometimes not containing the words of interest. Unfortunately, no sentences were identified that both contained the word *MVC* in the task phrase and were classified into one of the sub-categories of the “System” category. However, with improved accuracy, this question is believed to provide the user with useful information.

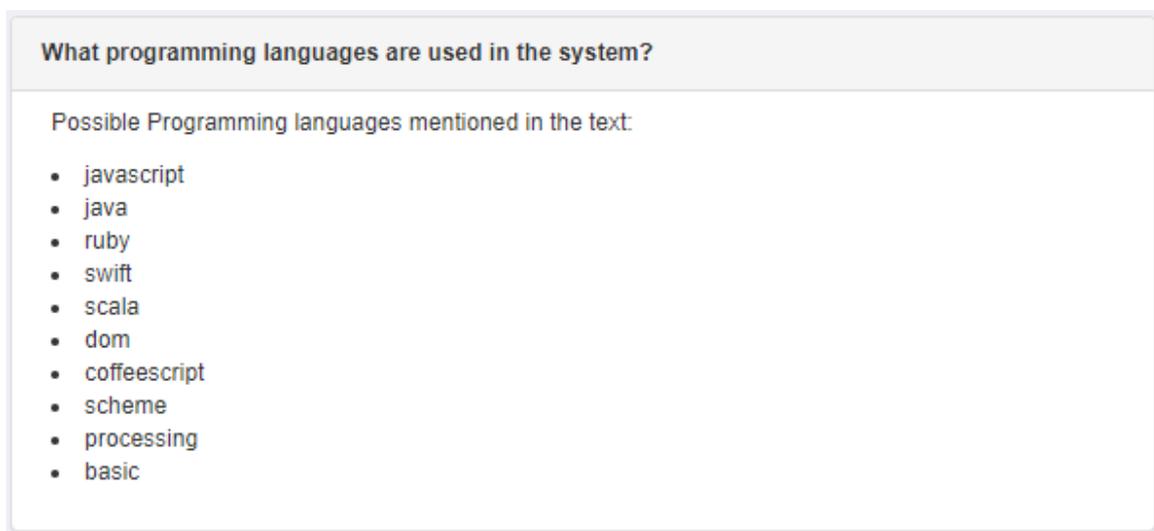


Figure 4.25: A screenshot showing results to Q7

How are the architecture patterns in the system implemented?

Possible Architecture patterns mentioned in the text:

- mvc
- state
- workflow
- composite

Categories shown in the graph:

Behaviour - technology solution Structure - technology solution Structure - technology solution composite

Words used to connect:

mvc state workflow composite

Result graph

Figure 4.26: A screenshot showing results to Q8

5

Results

This section presents the results of the classification and clustering algorithms, as well as both the quantitative and qualitative results of the system evaluation. This section therefore answers the two research questions: *How to automatically categorise natural language text presented in software documentation into software knowledge categories?* and *How to automatically identify relations between specific instances of knowledge in software documentation?*

5.1 Classification results

The classifiers were all configured to minimise overfitting, this can however be difficult when training data is limited, so in most cases, the models overfitted the training data to some extent. Overfitting under 15% was considered acceptable and the classifier with the highest validation F1 score that also had acceptable overfitting measures was chosen. However, minimising overfitting was given higher priority so if two models yielded similar F1 scores, but one had significantly less overfitting, the latter one was chosen. If classifiers had the same F1 score and the same amount of overfitting, cross-validation accuracy was used to choose one over the other.

An overview of the best results from both the flat and hierarchical approach can be seen in table 5.1.

Table 5.1: Classifier - hierarchy vs. flat

Section	Category	Flat		Hierarchy	
		Precision	Recall	Precision	Recall
Domain	Stakeholder	nan	nan	nan	nan
Requirement	Functional requirements & behaviour	47.59%	72.72%	88.88%	38.10%
	Non-functional requirements & behaviour	0.00%	0.00%	29.62%	57.14%
	General	0.00%	0.00%	88.88%	28.57%
	Use Case	100.00%	33.33%	88.88%	57.14%
	Feature	0.00%	0.00%	44.44%	57.14%

5. Results

System	Structure - Architecture	100.00%	16.67%	48.00%	28.80%
	Structure - Implementation - Technology Solution	37.50%	54.54%	36.92%	60.00%
	Structure - Implementation - Source Code	66.67%	50.00%	29.54%	72.00%
	Behaviour - Implementation - Tech Solution	100.00%	16.67%	45.00%	67.50%
	Behaviour - Implementation - General	nan	nan	nan	nan
	Data - Architecture	nan	nan	0.00%	0.00%
	UI design - Architecture	100.00%	25.00%	40.16%	73.63%
	UI design - Implementation	0.00%	0.00%	0.00%	0.00%
	Development Process	Development Practice	0.00%	0.00%	100.00%
QA - Identified issues		nan	nan	nan	nan
QA - Identified risks		nan	nan	0.00%	0.00%
QA - Testing		33.33%	50.00%	66.67%	20.00%
Document Organisation		60.00%	42.86%	33.33%	40.00%
Non-information		0.00%	0.00%	0.00%	0.00%
Uncertain		25.00%	66.67%	50.00%	14.29%

For the flat approach, the average precision is 35.63% and the average recall is 25.78%, while for the hierarchical approach, the average precision is 43.91% and the average recall is 35.24%. The hierarchical approach therefore yields considerably better results and is used to classify all sentences of the documentation.

The results per classifier for both the flat and hierarchical approach can be found in appendix A.

5.2 Clustering results

Clustering was used to identify relations between all sentences of the three categories; feature, use case and functional requirement and behaviour. Textual data is very high dimensional which can make it difficult to identify the correlations that hide within the data. Principal component analysis was therefore used to decrease the dimensionality of the data, and make it possible to explore the relationship between the features, functional requirements and use cases in the dataset.

Figure 5.1 shows the results from the principal component analysis where each dot represents one sentence from the feature, functional requirement behaviour or use case category.

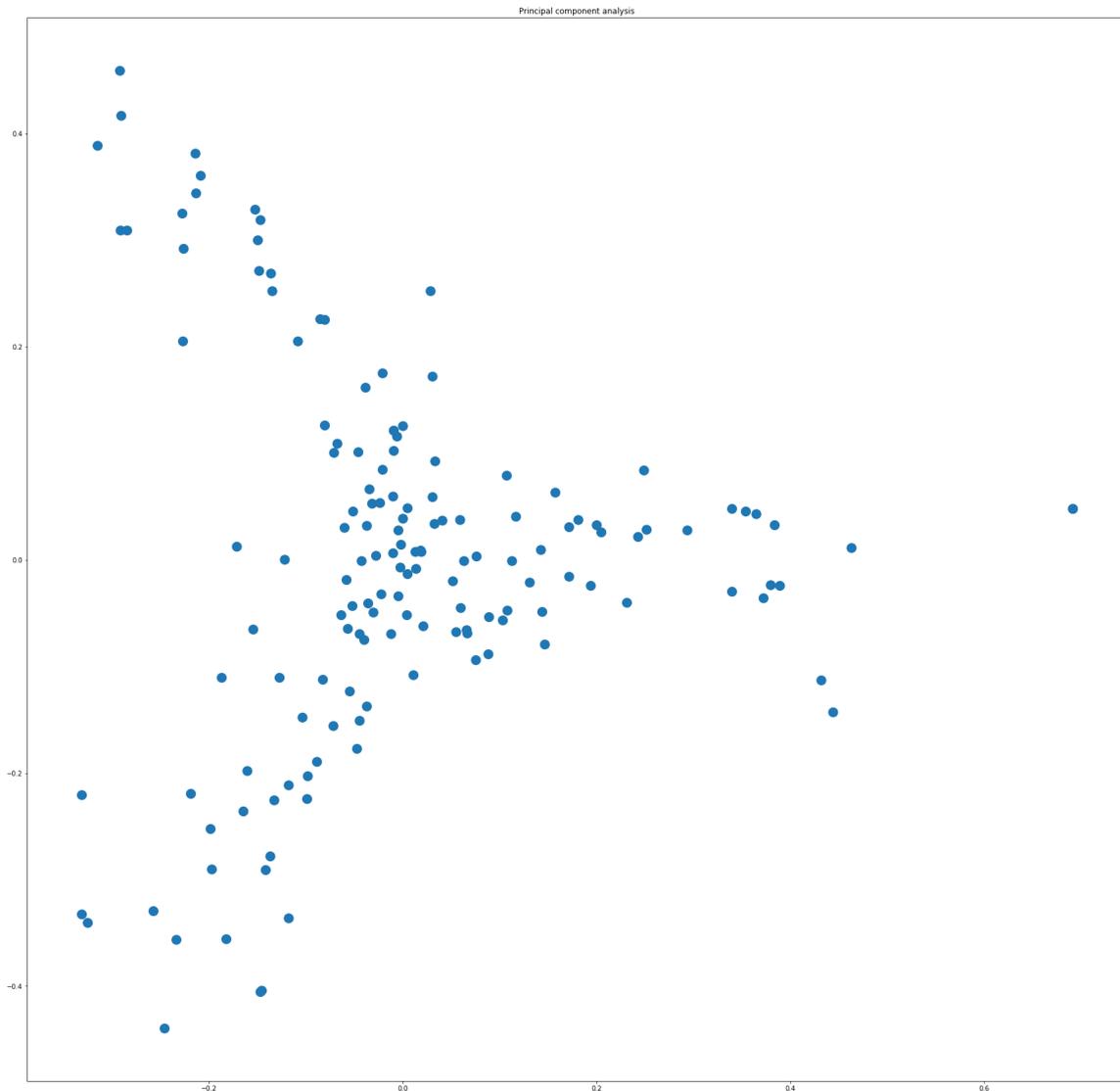


Figure 5.1: Results from the principal component analysis

5. Results

The system described in the documentation being analysed contains three features; user management, browse product, and purchase product, and the PCA results indeed show the data extending into three directions. In order to have something to compare to the clustering results, the sentences were labelled into three categories, based on which feature they relate to. The results from this manual labelling are shown in figure 5.2, where each category is represented by one color.

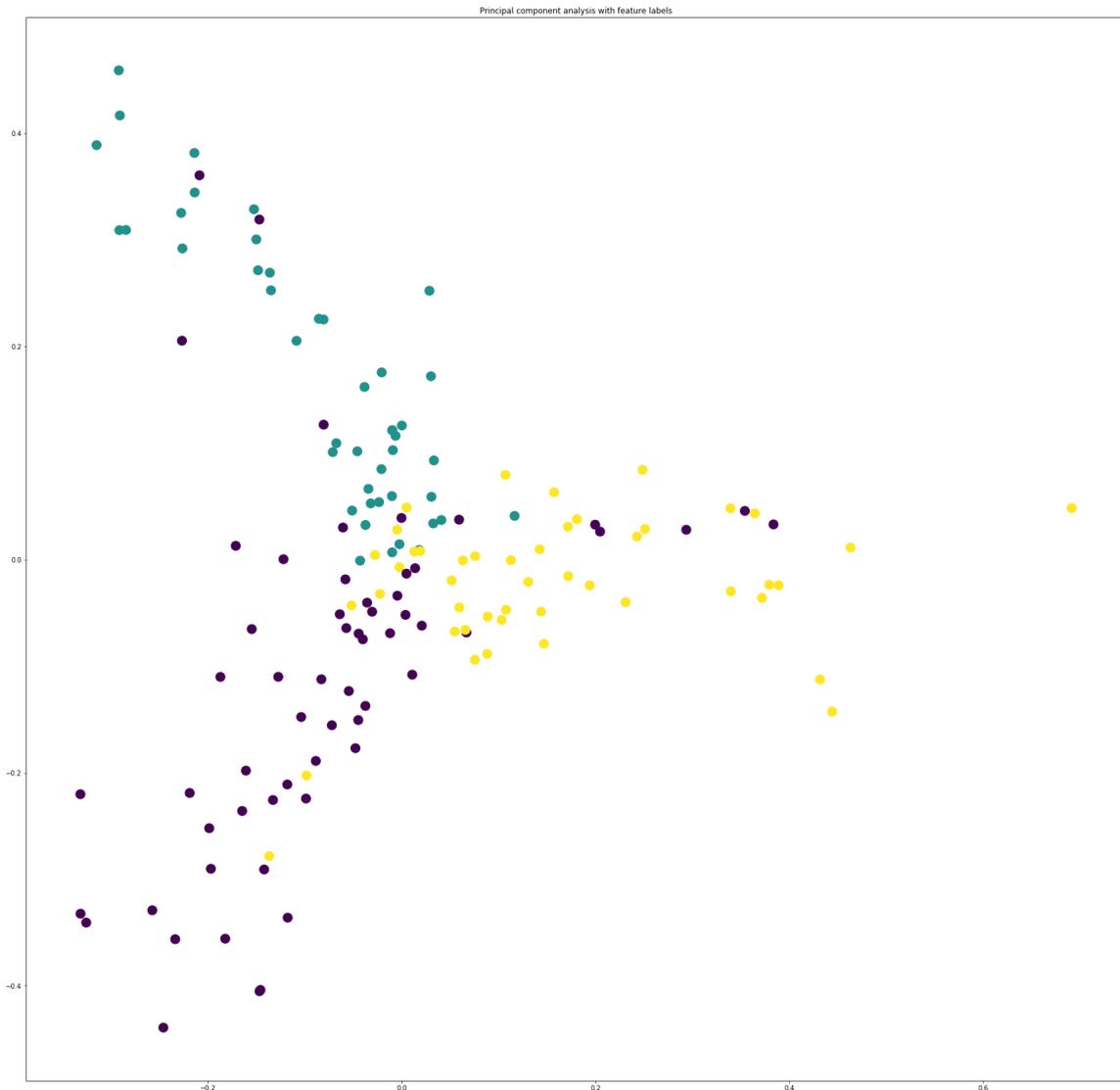


Figure 5.2: Manually labelled sentences

From this it is evident that each “arm” of the PCA results represents a feature quite accurately. A clustering algorithm was used to automatically identify the clusters. Both a K-means model and Gaussian-mixture model was used and they yielded the following results when compared to the manual labelling.

	K-means	Gaussian Mixture
User management Accuracy	44.44%	80.85%
Browse Products Accuracy	52.54%	77.78%
Purchase Products Accuracy	95.74%	76.27%
Total Accuracy	63.58%	78.15%

Table 5.2: Classifier - hierarchy vs. flat

The Gaussian-mixture model performed considerably better and also performed quite evenly across all clusters whereas the “purchase product” cluster in the K-means model was very dominant, resulting in high accuracy for one cluster but considerably lower accuracy for the other two clusters.

Figure 5.3 shows the results from the Gaussian-mixture model, where the data points that were clustered differently when compared to the manual labelling, were marked with red borders. A similar figure for the results from the K-means algorithm can be found in appendix B.

5.3 Results from system evaluation

Both quantitative and qualitative data was collected in the evaluation of the SDS. In the usability tests, the time it took participants to solve tasks was measured and the results are presented in section 5.3.1, feedback was then collected through the user experience interview and those results are presented in section 5.3.2.

5.3.1 Task times

All tasks in the usability tests were timed separately. The tasks to be solved with the documentation and with the SDS were very similar and were mapped to each other so it would be possible to compare task times for each task, in addition to the total task times. Measured task times can be seen in figure 5.4.

5. Results

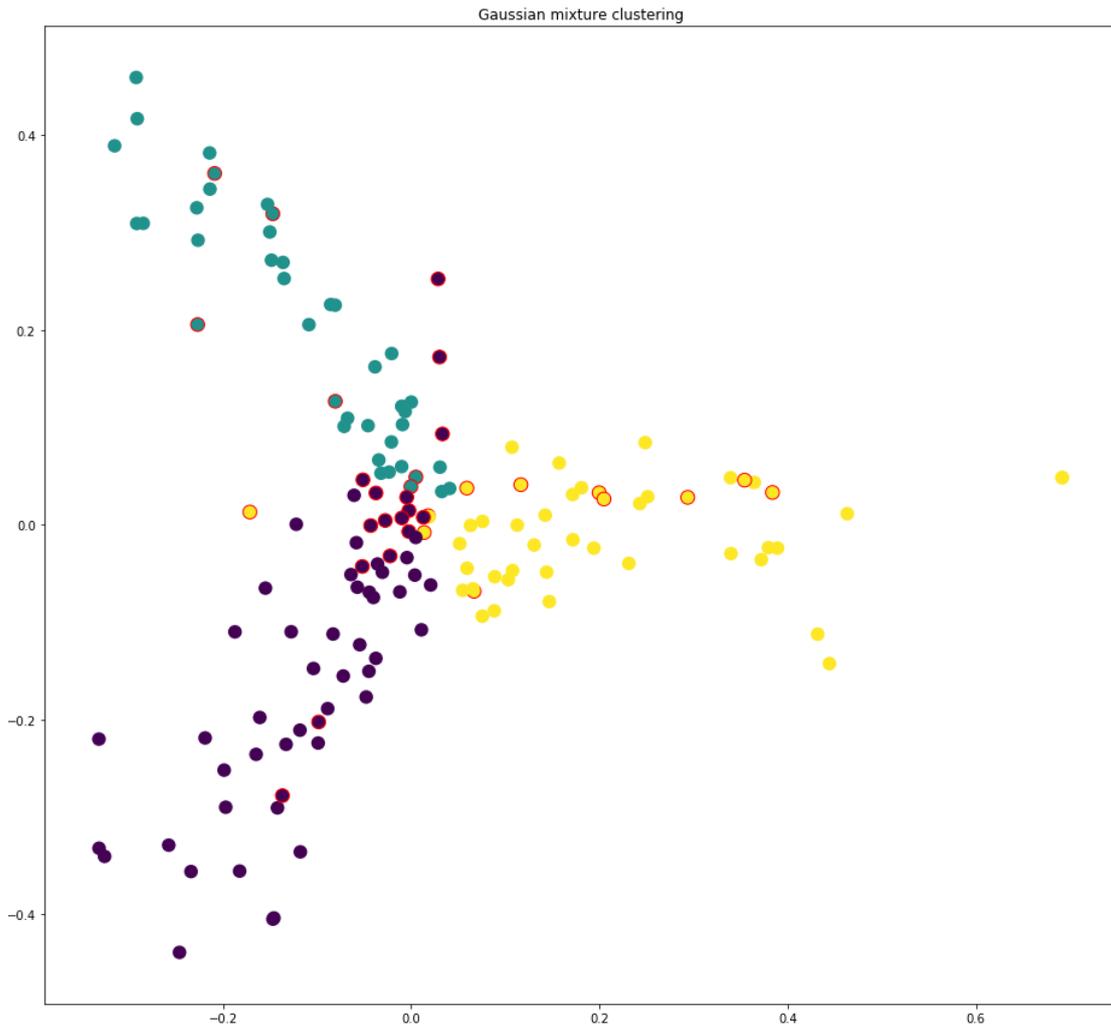


Figure 5.3: Clustering of feature, functional requirement and use case sentences, using the Gaussian-mixture model

Participant ID	Started with (Doc/system)	Documentation				Total	System				Total
		Task 1	Task 2	Task 3	Task 4		Task 1	Task 2	Task 3	Task 4	
P2	Documentation	223	258	409	128	1018	175	358	409	112	1054
P3	System	101	200	223	142	666	169	317	164	116	766
P4	System	197	124	319	204	844	438	238	548	100	1324
P5	Documentation	324	591	318	152	1385	252	283	292	113	940
P6	System	212	211	194	259	876	329	280	409	107	1125
P7	Documentation	424	702	603	216	1945	431	258	515	101	1305
P8	System	132	100	632	210	1074	333	540	589	243	1705
P9	Documentation	189	247	341	103	880	403	125	333	62	923

Figure 5.4: Task times for all tasks performed in the usability testing sessions

Task	Faster using documentation	Faster using SDS	Equal
1	5	3	
2	5	3	
3	2	5	1
4	1	7	

Figure 5.5: A count of how often participants were faster using documentation or SDS, per task

As can be seen in figure 5.4, only in 2 out of 8 times the participants were actually faster solving the tasks using the SDS. However, looking at certain tasks, it becomes clear that it was different between tasks whether participants performed better using the documentation or the SDS. Figure 5.5 summarises how often participants performed better using the documentation versus the SDS. For both tasks 1 and 2, 5 of 8 participants solved the tasks faster when using the documentation, whereas 3 participants were faster using the SDS. For task 3 and 4 however, participants were more often faster using the SDS, or in 5 of 8 times for task 3 and 7 of 8 times for task 4.

A t-test was used to check whether the results were significant. Before conducting the t-test, the data was checked for normality using the Shapiro-Wilk test and a QQ-plot, and the F-test was used to check whether the variance of the two data sets was equal. The results showed that both data sets follow normal distribution and have equal variance, these results can be found in appendix C. The following hypotheses were then set up for the t-test:

H0: Participants were as fast solving tasks using the documentation and the SDS.

H1: Participants were not as fast solving tasks using the documentation and the SDS.

The significance level was set to 0.05 and a two-tailed Welch t-test performed.

Average time using documentation	1045.33
Average time using SDS	1106.44
t-value	-0.3682
p-value	0.7178

Table 5.3: Results from the Welch t-test

The t-test shows that the probability of the participants being as fast solving the

tasks using the SDS and the documentation is 71.78%. It is therefore not possible to reject H0. **The results from the usability tests are therefore not significant and can not be used to state anything about whether the participants were faster or slower using the SDS, compared to the documentation.**

5.3.2 User experience

When participants had finished solving the tasks, they were asked to give feedback on the user experience of the system (see section 3.7 for more detailed information about the evaluation design).

Feedback on user experience was collected in two ways, firstly through the System Usability Scale questionnaire, and secondly, through four open-ended questions.

5.3.2.1 Results from System Usability Questionnaire

The System Usability Scale (SUS) is a standardised way to measure usability. It consists of 10 questions, each of which has 5 different response options; from strongly disagree to strongly agree.

Each participant marked the questions based on their feeling from the usability testing session. Each question's score is on a scale of 1-5. This scale was then translated to a 0-4 scale in accordance to SUS standards and finally each score was multiplied with 2.5 which resulted in each question score being on a scale from 0-10 and therefore the total SUS score being on a scale from 0-100.

Table 5.4 shows the score of each question of the SUS questionnaire, as well as the total SUS score.

Table 5.4: SUS score per question

Question	Score
I think that I would like to use this website frequently.	55.6
I found this website unnecessarily complex.	72.2
I thought this website was easy to use.	66.7
I think that I would need assistance to be able to use this website.	72.2
I found the various functions in this website were well integrated.	66.7
I thought there was too much inconsistency in this website.	72.2
I would imagine that most people would learn to use this website very quickly.	75.0
I found this website very cumbersome/awkward to use.	75.0
I felt very confident using this website.	47.2
I needed to learn a lot of things before I could get going with this website.	80.6
Total SUS score	68.3

5.3.2.2 Results from open-ended questions

In the open-ended questions, the goal was to give the participants freedom to express their feelings about the SDS. This allows for more honest and diverse feedback regarding user experience. The participants were asked to answer four open-ended questions. First, they were asked about what they liked in the system, second, they were asked how they felt when using the system, third, they were asked what improvements they would like to see in the system and finally, they had the opportunity to add any other comments.

Figure 5.1 shows a summarisation of the type of comments received when participants were asked what they liked about the SDS. Most participants, 8 of 9, mentioned that they liked the way of navigating the documentation in the SDS, this was also the type of comment that was most common. Good visualisation of information was the second most popular comment, ahead of the system's way of assisting user learning the documentation, but four different participants mentioned those two types of comments. Lastly, three participants liked the overview that the system provided, and finally one participants mentioned the improved traceability provided by the SDS.

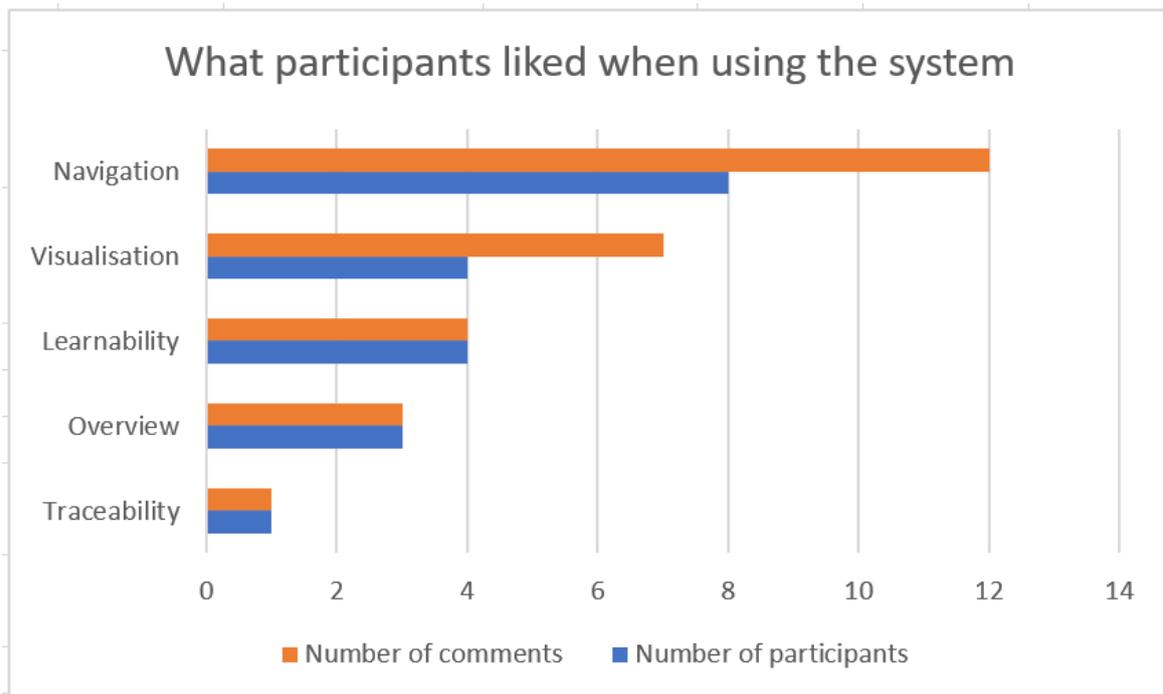


Figure 5.6: A summary of what participants liked about the SDS

Figure 5.2 shows a summarisation of how participants felt when using the SDS. Most commonly, participants felt supported by the system when solving the tasks. 3 participants felt that the system gave incomplete results. 2 different participants then gave three different comments about the system being interesting, but 2

5. Results

participants felt confused by it. Finally, 1 participant mentioned that he was more confident in his answers when using the SDS.

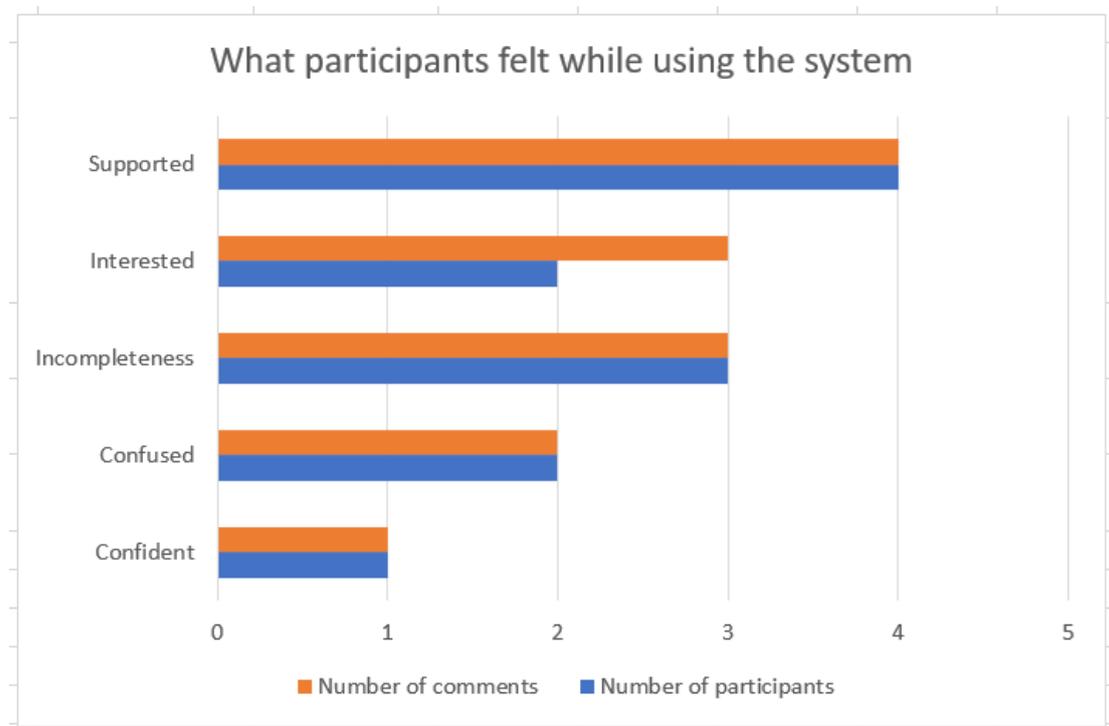


Figure 5.7: A summary of how participants felt when using the SDS

Figure 5.3 shows a summarisation of the type of improvement suggestion received from the participants. Most comments had to do with improvements of the result design and the UI design. Two participants then suggested making the drop-down lists and the results searchable, two participants suggested improving the machine learning algorithms and two participants made other feature suggestions.

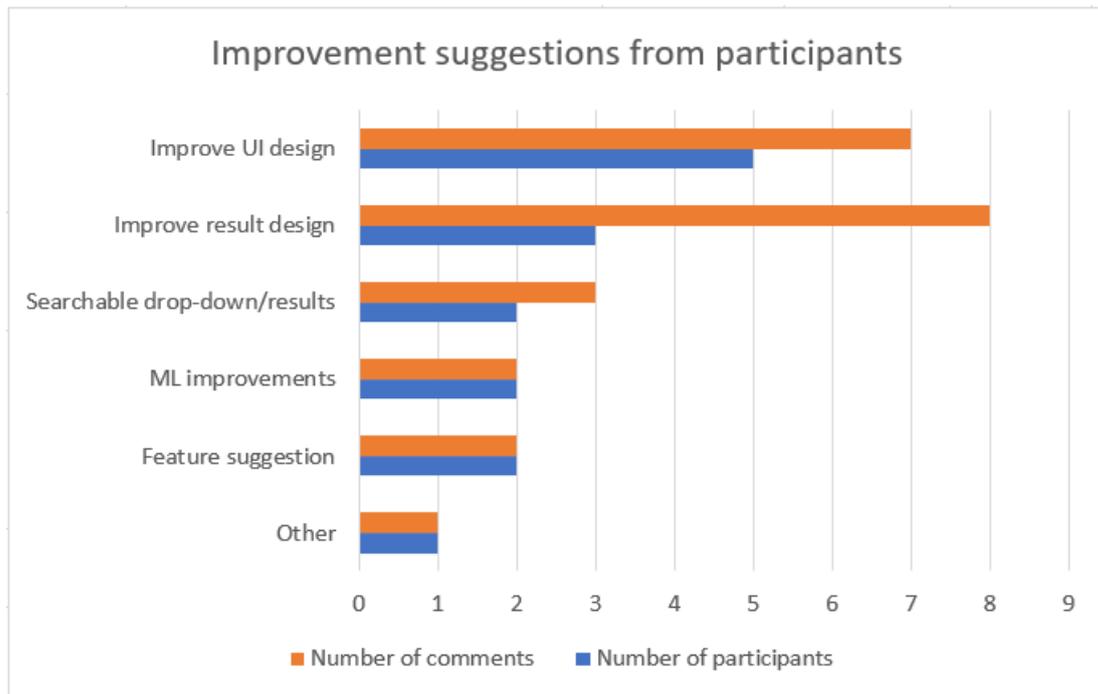


Figure 5.8: A summary of improvement suggestions for the SDS

The complete data collected during the user experience interview can be found in appendix F.

6

Discussion

In this section the main results of the research are further discussed and the threats to their validity identified and addressed.

6.1 Categories of software knowledge

To identify and define the categories of software knowledge used in this research, an iterative bottom-up approach. Snowflake, the documentation analysed in this research, is a very extensive software documentation and covers the requirement analysis, software design and architecture, the development process followed, and the system's quality assurance and testing, in high detail, and was therefore a good candidate. Realising the suitable abstract level for the categories was a challenge and defining the final categories took multiple rounds of iterations where a part of the documentation's sentences, or all sentences, were classified into the current prospect software knowledge categories.

The final categories were split into four high-level software knowledge categories; domain, requirement analysis, system, and development process. These four high-level categories are believed to be suitable for all software related knowledge contained in standard software documentation. There were also three other categories included; document organisation, non-information, and uncertain, that are needed for the knowledge that is contained in the software documentation but is not tightly related to the software itself. Those high-level categories then split further into sub-categories, forming a hierarchy. The sentences were categorised into the leaf-nodes of this hierarchy. The hierarchy has the benefits of making the categories easily extendable. As mentioned earlier, the categories were defined using a bottom-up approach and it is likely that in the future there will be a need to add more sub-categories to this classification structure. This hierarchy also enabled the use of a hierarchical classification approach when training a machine learning classifier to classify the sentences into one of the categories. This was especially useful as the training data was very limited which made it difficult for the classifier to recognise the difference between 25 different categories. Using the hierarchical approach allowed me to train several different classifiers, that only had to categorise the sentences into 2 to 7 categories at a time.

The requirement analysis and system categories were the largest high-level categories. The requirement analysis category had one level of sub-categories that

were inspired by a previous study by Tao et al. [11] and requirement analysis theory by Lauesen [32]. The system category was then split into 3 levels of sub-categories that were inspired by the 4+1 architectural view model by Kruchten [13]. Using current practice and theory was important when formulating the categories and their structure, as it increases the likelihood of the categories covering all knowledge contained in software documentation, leaves less room for misinterpretation, and will make future extensions easier.

Additionally, sentences that expressed a directive or a rationale were labelled. This information was not used in the research due to time limitations but might be valuable for future research. Information about rational and directives is, for example, important to answer questions about *why* some decision was made in the implementation process.

Although a traditional relational database would have been enough to fulfill the needs of this research, an ontology was chosen to store the sentences and their categories. Ontologies support automated reasoning about instances and have a good language for expressing instance relations, this can be a large benefit when storing software knowledge and relations. Ontologies are also easily extendable. Therefore, although the main advantages of the ontology were not utilised in this research, it might prove valuable for future studies that might extend this work.

Due to lack of data, it is difficult to draw conclusions about how good these knowledge categories are and how well they will generalise to other documentation. The accuracy of the classifier was rather low, but that was expected due to very little training data and therefore it is not possible to know whether some part of the inaccuracy was caused by the categories not being good enough. Also, it is not possible to say how well they will generalise to other documentation, as they were created using only one documentation. But as I have mentioned, measures were taken to minimise the effects of this by choosing a good case documentation, defining the categories with consideration to current practice and theory, and making sure they are clearly defined and easily extendable.

6.2 Using machine learning and natural language processing to identify software knowledge

The research questions referred to (1) how sentences could be automatically classified into software knowledge categories, and (2) how relations between these sentence could be automatically identified.

Lack of data had a great impact on the classification results. Implementing a good classifier for classifying natural text into categories that are very related to each other, like in our case where I am classifying sentences into 25 different categories of software knowledge, requires a lot of data, certainly much more than 800 instances. The classifier's precision and recall was therefore quite low, as

expected.

When data is scarce, overfitting and generalisability can be a large problem. When implementing the classifier, overfitting was minimised as much as possible, but often overfitting of about 15% had to be considered acceptable. Both this, and the fact that all training, testing and validation data came from the same and only documentation, makes it difficult to claim generalisability of the classifiers. If more data will be accessible for a study similar to this in the future, it is unlikely that the same classifiers will prove to perform best.

Two different methods were used to identify relations between instances, clustering and connecting sentences' task phrases.

Clustering is only a viable solution to identify relations when all instances belong to one of the potential clusters. This is often not the case in software documentation where there are many independent sentences that do not necessarily relate to anything elsewhere in the text, and defining a certain amount of clusters can be difficult. However, in this research I identified one case where clustering is very useful, that is when identifying what feature each functional requirement and use case belongs to. The most difficult and limiting factor in a research like this is the data annotation, and thus an unsupervised method like clustering, that does not require annotated data, is incredibly beneficial. This solution however has the drawback of not being fully automated; a manual input was required for the number of clusters, k . The Snowflake documentation declares that the system has 3 features and thus k was manually set as 3. There do exist some algorithms that are supposed to detect k automatically, like for example the G-means algorithm by Hamerly et al. [33] and the split and merge k-means approach by Muhr et al. [34]. These algorithms however often do not perform well and automatically identifying k is still a large challenge when using clustering methods. On top of that, working with natural language makes this an even larger challenge, as there is a lot of noise present in the natural language data.

The second method used to identify instance relations was extracting task phrases and connecting sentences that had some words in common in their task phrases. This method does therefore not utilise any machine learning methods, and has the benefit of not requiring any annotated data. The drawback of this method is that it does introduce a lot of noise to the results, as not all words in the task phrases are necessarily tightly related to the information that the user is looking for. For example, the task phrase of the sentence "*As a customer, I want to list all products of the shop*" is "*List products of shop*". In this case, the user is most likely looking for more information about the *listing* functionality of the system, however, the system shows sentences connected to *list*, *product* and *shop* (the word *of* is excluded since it is a stop word). In order to allow the user to minimise noise like this in the results, all task phrase words were displayed as filters in the system, so in this case the user could filter out the results caused by the task phrase words

product and *shop*. As the qualitative feedback from the usability testing showed that users were sometimes quite overwhelmed with the amount of results shown by the system. Like I discuss in sections 6.3 and 6.4, I believe the training that participants received in the usability testing sessions was too limited, and with more experience, I do believe that users would not be as overwhelmed with the results. Nonetheless, this is definitely a part of the SDS that can be improved with further research.

Identifying technology concepts allows the SDS to answer questions about concepts that are used to implement the system, and even how they are implemented, if combined with the Task Phrase Extractor. Potentially, the system could also answer questions about why certain, e.g. architecture patterns or programming languages, were chosen over other possibilities, by implementing the additional functionality of being able to recognise rationale and directives. The technology concepts were identified by finding matches between the documentation text and a list of technology concepts from the Witt database. This method has the drawback of being static, so the database might not contain technologies invented after the database was created. However, machine learning techniques were used to create the database and therefore it has the potential of being dynamic, that is, the machine learning algorithm could be used to update the list of concepts regularly.

6.3 Usability tests

As can be seen in the result section (section 5.3.1), the participants were on average slower solving the tasks when using the SDS. However, the results from the two sided t-test showed that the task time results were not significant and thus it is not possible to conclude that using the SDS affects the time it takes to solve tasks, compared to using the documentation itself. This is caused by the difference between task times when using documentation versus using the SDS being too little to be identified with such few participants. Given the data collected from the 8 participants we had, roughly 230 participants would have been needed to get significant results, for a statistical significance threshold of 0.05.

I believe the reason why the participants did not perform as quick as was expected beforehand is mainly because participants only received about 10-15 minutes of training before being asked to solve quite complex tasks using the system. It was apparent in the evaluation sessions that more training was required for users to efficiently use the system to solve extensive tasks.

Task 1	Faster using documentation	Faster using system
Starts with documentation	1	3
Starts with system	4	0

Task 3	Faster using documentation	Faster using system
Starts with documentation	0	3
Starts with system	2	2

Task 2	Faster using documentation	Faster using system
Starts with documentation	1	3
Starts with system	4	0

Task 4	Faster using documentation	Faster using system
Starts with documentation	0	4
Starts with system	1	3

Figure 6.1: Task times for all tasks performed in the usability testing sessions

Figure 6.1 shows if participants started by solving the documentation tasks or the system tasks and when they were faster. Looking more closely into the results of tasks 1 and 2 shows that the learning bias for those tasks was quite extensive. For both tasks 1 and 2, 7 of 8 participants were faster solving the task the second time, so if they started by solving the documentation tasks, they were faster solving the tasks using the system, and vice versa. Looking at the results in figure 5.4 it is also apparent that the learning bias is in many cases very large, for example, participant P7 started by solving tasks using the documentation and it took him/her 702 seconds to solve task 2 using the documentation, but when solving a similar task using the system he improved his/her time by 444 seconds, or about 7.4 minutes. The difference was then often similar in the other direction, when participants started solving tasks using the SDS and then improved a lot when they solved a similar task using the documentation. I did try to minimise the effect of the learning bias by making half of the participants start solving tasks using the documentation and the other half using the SDS, however, when the bias is this large, it affects the overall results considerably.

Even though the usability tests do not yield significant results and had some drawbacks like the learning bias I discussed, they do indicate that further improvements need to be made to the SDS in order to develop it into a tool that will help users to effectively navigate software documentation. On top of that it is also clear that even though the system itself is simple, using it can be complex for users who are unfamiliar with it. More extensive training sessions before performing usability tests on a system like the SDS would therefore be crucial to collect as good data as possible. A more detailed discussion of potential improvements to the SDS can be found in section 6.4.

6.4 User experience

When participants had finished solving the tasks for the usability testing, they were asked to answer questions about their user experience.

In a study from 2011 by Sauro [31], where data was collected and analysed from over 5000 different users in 500 different evaluations, the average total SUS score was 68. The SDS' final SUS score was 68.33, so it scored slightly above average. There were two questions that scored below 60, the question *I felt very confident using this website* scored the lowest, 47.22. I believe there are mainly two reasons for this. Firstly, again the very limited training that the participants received before solving the tasks. Secondly, I believe that the low accuracy of the machine learning results affected the confidence of the participants. Seeing false positives within the results, or realising that something is missing from the results, has considerable effects on the user as they tend to start doubting all results. This then is likely to have affected the results of the first question, *I think that I would like to use this website frequently*. Not feeling confident about the information that the system is providing you is likely to be a large deciding factor for the user. Then there were 3 questions that scored 75 or higher. Two of those questions were about the system's learnability. This might sound like a contradiction to what I have been saying, that the users would need more training to be able to efficiently and correctly carry out complex task in the system. However, I believe the reason for this is that the user interface itself is very simple and the whole system is on one page in a web system. But even though the system itself is not complex and it does not have many functionalities, the complex part is to translate the task you have to one of the questions offered by the system. This I believe takes more than 10 minutes to get a feeling for. The third question that scored high was *I found this website very cumbersome/awkward to use*. I believe this question scored high because the user interface is simple. There are few functionalities and all of them are performed from the front page, which is the website's only page.

Through the open-ended questions, it was possible to give the participants more freedom and to get feedback without introducing bias. The questions were supposed to affect the answers as little as possible and were therefore phrased in a general way to reduce the likelihood of them introducing bias. The moderator also made sure to participate as little as possible in the discussion while the participant was answering the questions, again to decrease the likelihood of introducing bias. The results show that most participants felt the system helped them when navigating the system. Many then liked the visualisation provided by the system and thought it helped them get an overview and an understanding of the contents of the documentation that was being analysed.

When asking the participants how they felt when using the SDS, many mentioned that they felt supported, which likely goes hand in hand with the fact that the system was helping them navigate and understand the documentation. However, some participants also mentioned that they felt incompleteness in the system and were confused. This likely relates to the confidence issue that was apparent in the SUS questionnaire results and has already been discussed.

When asking participants about improvements they would like to see in the SDS, many mentioned minor user interface details like, for example, changing the

filter button design and add hierarchy to the documentation text. Many suggestions were also given about possible improvements to the result design itself, like for example giving a better overview of where the results are in the documentation, and a more detailed categorisation/clustering of the results in the graph. Some participants mentioned that it would be good to make the user story selection and the results searchable to improve the navigation within the SDS. Finally, participants also mentioned the importance of improving the accuracy of the automatic results.

I believe these improvement suggestions support the analysis of the SUS results. The amount of suggestions related to navigation of the results and result related UI design is likely triggered by users having difficulties with getting an overview of the results displayed by the system. The improvement suggestions are all very good, but the need for some might be eliminated by making the results more accurate and more focused.

In order to improve the SUS score and the overall user experience of the SDS, I believe it is most important to improve the accuracy of the automatic extraction methods. When the results of the system are reliable, the user's confidence will increase, and user confidence is obviously essential for developing a good system. Also, I believe it might be necessary to make the system's results more focused. There are several things that need to be considered in regards to that. First, it might be necessary to make the questions in the system more focused. In order to do this, some research would have to be made to better figure out what questions are most important to the user. Second, increasing the result accuracy will lead to more relevant and focused answers. In this case, the classification accuracy, Task Phrase Extractor connections and technical concept identification should all be investigated to figure out how noise can be reduced.

6.5 Threats to validity

This section describes the threats to validity of this research, and how these threats were addressed.

6.5.1 Internal Validity

Firstly, a threat to internal validity is the bias in the data annotation, since all training data was annotated by me. For the sentence classification annotation, I got three other persons to go over different parts of the annotation, two of those were completely unrelated to the study and should not be biased in any way. For the cluster labels that I created to compare to the GMM clustering results, I then minimised the bias by labelling the data before I created the clustering algorithm. I therefore did not know the true results when I labelled the data and should therefore not be biased.

The results of the usability testing did not show with significance any differ-

ence in the time that it took participants to solve tasks using the documentation versus using the SDS. However, as in most evaluations, there are some internal validity threats that are worth discussing.

When it came to the usability testing sessions there were two limitations that may have caused bias in the results, time and the fact that not all sessions were run by the same moderator.

The time of the sessions was limited since the participants were volunteers that did not get any compensation for their time. I therefore wanted to take up as little of their time as I could without highly affecting the results of the usability tests. Long usability testing sessions are also more likely to tire out the participants, affecting the quality of the results. In order to mitigate this threat, participants were allowed to ask for pointers if they felt lost when solving the tasks. The pointers given did not contain any information about the solution to the task, but either further explained the task itself, or gave information about e.g. the structure of the documentation text or how some functionality works in the SDS. During the usability testing sessions, it varied how many pointers the participants asked for, some asked quite often while others never asked for pointers. Most pointers given were only to clarify the meaning of the tasks. A list of all pointers given during the usability sessions can be found in appendix G. These pointers may have caused bias in the results, but I minimised that by standardising the procedure as much as possible. No pointers were given without the participant asking for them, but when they had spent 5 minutes on the same task without asking for a pointer, they were gently reminded that they were allowed to ask for those pointers. This also applied both when participants were solving tasks using the documentation and the SDS. Secondly, as I already said, the pointers given never directly related to the answer to the task and most pointers given were to clarify the task they were supposed to solve. I therefore believe that the pointers did not have a significant effect on the result of either the documentation times or the SDS times.

The other issue with the sessions was that due to my location, I was not able to moderate all the sessions. Three sessions were therefore run by another moderator. In order to minimise the effects of this, the sessions were standardised as much as possible. A welcoming text was written so that all participants would receive the same information about the session, a tutorial video was made so that all participants would receive the same training, and the substitute moderator made sure to practice and fully understand the system so he could provide the same level of help as me.

The selection of the participants is then another threat to internal validity. It was not possible to select a random sample from the domain of all software developers, instead I had to contact people that I knew had background in software development, and might be willing to participate. This can cause bias both because the group of participants might not be representative of all software developers that would benefit from using a system like the SDS (for example, the participants

might be very familiar with using documentation or might not be familiar with using documentation at all), and because I might be biased in my selection of prospect participants. These biases are difficult to mitigate, but efforts were put into minimising their effects as much as possible. Firstly, I made sure that the participants all had background in software engineering, but tried to make sure that the background was as diverse as possible. The participants were therefore from different levels of academia, and industry as well. All participants had some level of experience with requirement analysis and software architecture. However, I was not able to fully balance participants from academia and industry, as I was only able to get one participant with a background in industry. In order to minimise the bias caused by me selecting the participants, I tried to get as many suggestions from others as possible and sent invitations to all participants that were suggested. Everyone that responded was then included in the evaluation.

6.5.2 External Validity

Given the very limited data used in this research, generalisability was never its main purpose.

The SDS is developed using only one case documentation, Snowflake. Even though this documentation was carefully chosen, it can not be representative of all software documentation. This affects the generalisability of both the software knowledge categories defined and the classifier that was trained to recognise those categories.

The SDS can then certainly not answer all questions that might come to a developer's mind when looking for information in software documentation. The tasks of the usability tests were chosen to showcase the type of tasks that can be solved with the help of the SDS. The four tasks of the usability tests therefore do not represent all tasks that developers normally solve using documentation. This affects the generalisability of the usability testing results. In its current state, the SDS will likely not be faster at solving all tasks that might come up when developers are using documentation. However, the current SDS is a proof-of-concept and is mainly supposed to showcase the potential of such a system.

The sample size of the usability test is also only 8, which is very small and means that it is not possible to generalise the results to the domain of all software engineers.

As I said, generalisability was not a primary concern of this research. The main goal was to create a proof-of-concept system that would showcase the possibilities of using machine learning and other automatic methods to extract knowledge from software documentation that could be used to support the user when navigating those large documentations.

6.5.3 Construct Validity

The main threat to construct validity is the mono-method bias, as only one method of measurement was used during the usability tests. To mitigate this bias, both quantitative and qualitative feedback was collected during the usability tests. The quantitative results are the task time measured, and the qualitative results is the feedback collected through both the SUS questionnaire and the open-ended questions of the user experience interview. This made it possible to compare and analyse the results to see how they align.

Another threat to construct validity was then the learning bias imposed on the usability testing, as the users always solved two very similar tasks, one using the documentation and one the SDS. The task that the user solves secondly, is therefore always somewhat biased, as the user has experience from solving a very similar task earlier. In order to mitigate this threat, half of the participants started by solving tasks using the documentation and the other half started by solving tasks using the SDS. However, as already discussed, for the first two tasks, the learning bias was greater than expected and is suspected to have a considerable impact on the total results.

7

Conclusion

This research contributes to the OD3 vision of future software documentation and focuses on the challenge of using automatic methods, such as machine learning and natural language processing, to extracting knowledge from documentation. In this research I (1) identified categories of software knowledge that can be recognised in software documentation, (2) automatically classified the documentation's sentences into those categories of software knowledge, and (3) automatically identified the relations between different sentences of the documentation. To achieve this, one documentation was analysed and a web system called the Software Documentation Supporter (SDS), was developed to showcase and evaluate the results.

The software knowledge categories were identified by using an iterative bottom-up approach to identify the software knowledge categories and the appropriate level of abstraction to work with. As only one documentation was used to identify those categories, it might be necessary to add more to generalise to all software documentation, but measures were taken to make future extensions as easy as possible.

The sentence classification was achieved by training a hierarchical classifier that classifies all sentences into one of the 25 defined software knowledge categories. As training data was very limited, the precision and recall of the classifier was only 35.63% and 25.78%, respectively. But by gathering more training data, these numbers are expected to improve considerably. The trained classifier is not expected to generalise well, again because the training data was incredibly limited and some overfitting had to be accepted.

The identification of sentence relations was solved by using two methods; clustering and connecting the words of the task phrase extracted from each sentence. Firstly, clustering was applied to the features, functional requirements and use cases, to identify which functional requirements and use cases belonged to which features. This allowed the system to give a good overview of the requirement analysis of the system. Secondly, task phrases have been shown to be useful when extracted from natural text, as they contain verbs and associated objects which, in the domain of software documentation, often contain the most important knowledge presented by a certain sentence. The extracted task phrases were then used to connect the sentences of the documentation that had some words in common in their task phrases.

SDS is the system I developed to showcase and evaluate the results. The SDS shows the complete documentation text, but also has features to help you navigate the text. In the SDS, users can ask questions about the documentation they are analysing and receive answers in the form of natural language and a graph. The graph shows the sentences that are believed to contain relevant knowledge and how they relate to each other. The relevant sentences are all highlighted in the documentation and all sentences in the graph are clickable to take you to the correct place in the documentation.

Evaluations were then performed on the SDS. The evaluation was twofold, first usability tests were performed with a sample size of 8 participants, and second, a qualitative interview with a sample size of 9 participants was used to collect feedback about the user experience of the SDS. The usability tests did not show with significance any difference in the amount of time it took participants to solve tasks with or without the help of the SDS. However, the user experience interview showed that participants did feel supported when using the system and liked that it helped them navigate through the software documentation, and as stated in section 1.3, the purpose of this study was to use automatic techniques to extract knowledge from software documentation and to use this knowledge to help users navigate through documentation to find their desired information faster.

Future work should include gathering more data from software documentation to increase generalisability of the defined software knowledge categories. More annotated data is also required to increase the generalisability and the precision and recall of the classification results. Another interesting task would also be to look further into the possibilities provided by the Task Phrase Extractor, as the current implementation produces a lot of noise. Other interesting areas to focus could be (1) handling custom queries from the user, instead of presenting him with a list of predefined questions, (2) researching the most effective presentation of knowledge extracted from software documentation, (3) some integration of the SDS and SEC, the tool developed by Tao et al. [11] and (4) extending this research to include images and source code.

Bibliography

- [1] Nosál', M. & Porubän, J. (2016) Preliminary Report on Empirical Study of Repeated Fragments in Internal Documentation. *Proceedings of the Federated Conference on Computer Science and Information Systems, 8*, 1473-1576. doi: 10.15439/2016F524
- [2] Vranić, V. Porubän, J., Bystrický, M., Frtala, T., Polášek, I., Nosál, M., Lang, J. (2015) Challenges in preserving intent comprehensibility in software. *Acta Polytechnica Hungarica, 12*(7), 57-75. doi: 10.12700/aph.12.7.2015.7.4.
- [3] Luciv, D.V., Koznov, D.V., Chernishev, G.A., Terekhov, A.N. (2018) Detecting Near Duplicates in Software Documentation. *Programming and Computer Software, 44*(5), 335-343. doi: 10.1134/S0361768818050079.
- [4] Robillard, M.P., Marcus, A., Treude, C., Bavota, G., Chaparro O., Ernst, N., Gerosa, M.A., Godfrey, M., Lanza, M., Linares-Vásques, M., Murphy, G.C., Moreno, L., Shepherd, D., Wong, E. (2017) On-Demand Developer Documentation. *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 479-483.
- [5] Nassif, M., Treude, C., Robillard, M. (2018) Automatically Categorizing Software Technologies *IEEE Transactions on Software Engineering*, 1-1. doi10.1109/TSE.2018.2836450
- [6] Escoriza, L.L. (2014). Analysis , design and development of a web-shop template using SPHERE.IO e-commerce platform. (Unpublished master's thesis). Universitat Politècnica de Catalunya.
- [7] Xia, X., Bao, L., Lo, D., Xing, Z., Hassan, A.E., Ahmed, E., Li, S. (2018) Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering, 44*(19), 951-976.
- [8] Lethbridge, T.C., Singer, J., Forward, A. (2003). How software engineers use documentation: the state of the practice. *IEEE Software, 20*(6), 35-39. doi: 10.1109/MS.2003.1241364.
- [9] Robillard, P. Uddin, G. (2015). How API documentation fails. *IEEE Software, 32*(4), 68-75. doi: 10.1109/MS.2014.80.
- [10] Bass, L., Clements, P., Kazman, R. (2012). *Software Architecture in Practice*. 3rd ed., Addison-Wesley.
- [11] Tao, A. Roodbari, M. (2018). *Towards automatically generating explanations of software systems* (Unpublished master's thesis). Chalmers University of Technology, Gothenburg, Sweden.
- [12] Gruber, T.R. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition, 5*(2), 199-220.
- [13] Kruchten, P. (1995). Architectural Blueprints - The "4+1" View Model of Soft-

- ware Architecture *IEEE Software*, 12(6), 42-50.
- [14] P. Mayring. (2014). Qualitative Content Analysis. Theoretical Foundation, Basic Procedures and Software Solution. Klagenfurt.
- [15] Peffers, K., Tuunanen T., Rothenberger, M.R., Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, 24(3), 45-77.
- [16] IEEE Recommended Practice for Software Requirements Specifications (2009) IEEE Std 830-1998(R2009)
- [17] Systems and software engineering - Architecture description. (2011) ISO/IEC/IEEE 42010:2011(E).
- [18] IEEE Standard for Software and System Test Documentation (2008) IEEE Std 829-2008
- [19] Soliman, M. (2018) Acquiring Architecture Knowledge for Technology Design Decisions (Doctorate thesis). University of Hamburg. Hamburg, Germany.
- [20] Tang, A., Bi, T., Liang, P., Yang, C. (2018) A Systematic Mapping Study on Text Analysis Techniques in Software Architecture. *Journal of Systems and Software*, 533-558.
- [21] López C., Codocedo V., Astudillo H., Cysneiros L. M. (2012) Bridging the gap between software architecture rationale formalisms and actual architecture documents: an ontology-driven approach. *Science of Computer Programming*, 66-80.
- [22] Anvaari M., Zimmermann O. (2014) Semi-automated design guidance enhancer (SADGE): a framework for architectural guidance development. *Proceedings of the 8th European Conference on Software Architecture (ECSA)*, 41-49.
- [23] Nicoletti, M. Díaz-Pace J.A., Schiaffino S. (2011) Towards software architecture documents matching stakeholders' interests. Proceedings of the 2nd International Conference on Advances in New Technologies, Interactive Interfaces, and Communicability (ADNTIIC), 176-185.
- [24] Ko, A. J., Myers, B. A., Coblenz, M. J., Aung, H. H. (2006) An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *IEEE Transactions on Software Engineering*, 32(12), 971-987.
- [25] Silito, J., Murphy, G. C., Volder, K. D. (2008) Asking and Answering Questions during a Programming Change Task. *IEEE Transactions on Software Engineering*, 34(4), 434-451.
- [26] Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc.
- [27] Pedregosa, F. et al (2011) Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.
- [28] Brooke, J. (1996). SUS: a "quick and dirty" usability scale. Usability evaluation in industry (P. W. Jordan, B. Thomas, B. A. Weerdmeester, A. L. McClelland (Eds.)), 189-194. London: Taylor and Francis.
- [29] Bangor, A., Kortum, P.T. and Miller, J.T. (2008) An Empirical Evaluation of the System Usability Scale *International Journal of Human-Computer Interaction*, 24(6), 574-594. DOI: 10.1080/10447310802205776
- [30] Treude, C., Robillard, M.P., Barthélémy, D. (2015) Extracting Development

- Tasks to Navigate Software Documentation. *IEEE Transactions on Software Engineering*, 41(6), 565-581.
- [31] Sauro, J. (2011) A Practical Guide to the Systems Usability Scale. *CreateSpace Independent Publishing Platform*.
- [32] Lauesen, S. (2002) Software Requirements, Styles and Techniques. *Addison-Wesley*. ISBN: 0 201 74570 4.
- [33] Hamerly, G., Elkan, C. (2002) Learning the k in k-means. University of California, San Diego.
- [34] Muhr, M., Granitzer, M. (2009) Automatic Cluster Number Selection Using a Split and Merge K-Means Approach *International Workshop on Database and Expert Systems Application*. DOI:10.1109/DEXA.2009.39

A

Appendix A

Here, all results from the sentence classification are presented. Each table represents one classifier and shows the results from each model that was tested. The model that was used is highlighted with orange.

Table A.1 shows the results from the flat approach. Tables A.2 to A.10 then show the results from all levels of the hierarchical approach.

Table A.1: Flat approach - classifier results

Classification Model	CV1		CV2		CV3		CV4		CV5		CV		Training Testing Overfitting		Validation	
	accuracy [%]	average accuracy [%]	recall [%]	recall [%]	Precision [%]	Recall [%]	F1									
BernoulliNB(alpha=2)	30.22	28.81	29.82	32.35	28.66	29.97	37.59	25.93	11.66	23.11	23.71	23.41				
ExtraTreesClassifier(max_features=500, class_weight='balanced', min_impurity_split=0.9)	34.62	32.77	30.41	35.93	32.32	33.21	46.35	31.48	14.87	27.14	25.77	26.44				
KNeighborsClassifier(n_neighbors=100)	40.11	45.20	40.36	43.11	42.68	42.29	44.34	40.28	4.06	52.15	38.14	44.06				
LinearSVC(C=0.01, class_weight='balanced', dual=True, fit_intercept=True, loss='squared_hinge', multi_class='ovr', penalty='l2')	40.66	38.98	43.27	46.71	39.63	41.85	69.53	49.07	20.46	37.92	32.99	35.28				
LogisticRegression(n_jobs=1, multi_class='multinomial', class_weight='balanced', solver='lbfgs', C=0.03, penalty='l2')	37.36	37.85	36.26	43.71	34.76	37.99	60.58	43.06	17.52	30.60	24.74	27.36				
NearestCentroid()	49.45	44.07	48.54	56.89	43.90	48.57	82.66	53.70	28.96	45.93	40.21	42.88				
RandomForestClassifier(class_weight='balanced', min_impurity_split=0, max_depth=4, random_state=0)	14.29	19.77	22.22	14.97	15.24	17.30	33.03	25.93	7.10	33.55	16.49	22.11				
SGDClassifier(alpha=0.5, loss='log', penalty='l2', class_weight='balanced', random_state=42)	11.54	3.39	25.15	21.56	9.76	14.28	42.70	30.09	12.61	21.01	24.74	22.72				

Table A.2: Level 1

Classification Model	CV1		CV2		CV3		CV4		CV5		CV		Training Testing Overfitting		Validation	
	accuracy [%]	average accuracy [%]	recall [%]	recall [%]	Precision [%]	Recall [%]	F1									
BernoulliNB(alpha=1, binarize=0, fit_prior=True, class_prior=None)	61.93	62.21	62.57	64.91	60.23	62.37	75.86	62.96	12.90	43.79	47.69	45.66				
ExtraTreesClassifier(max_features=500, class_weight='balanced', min_impurity_split=0.8)	30.68	34.88	35.09	31.58	32.16	32.88	37.07	28.24	8.83	61.66	30.77	41.05				
KNeighborsClassifier(n_neighbors=100)	54.54	56.40	59.06	52.05	56.72	55.76	55.17	53.24	1.93	46.02	55.38	50.27				
LinearSVC(C=0.01, class_weight='balanced', dual=True, fit_intercept=True, intercept_scaling=1, loss='squared_hinge', max_iter=500, multi_class='ovr', penalty='l2', tol=1e-05)	59.66	56.70	63.16	61.40	57.31	59.70	68.27	56.48	11.79	62.48	60.00	61.21				
LogisticRegression(n_jobs=1, multi_class='multinomial', class_weight='balanced', solver='lbfgs', C=0.1, penalty='l2')	61.21	59.30	60.82	63.74	60.23	61.21	76.38	59.26	17.12	63.54	56.92	60.05				
NearestCentroid()	59.09	55.81	66.10	63.16	56.14	60.57	78.97	61.11	17.86	52.74	49.23	50.92				
RandomForestClassifier(class_weight='balanced', min_impurity_split=0, max_depth=4, random_state=0)	42.05	34.88	28.07	47.37	30.41	36.56	52.76	41.67	11.09	19.29	23.77	21.30				
SGDClassifier(alpha=0.01, loss='log', penalty='l2', class_weight='balanced', n_iter_no_change=3, early_stopping=True) #63,5 - 63	59.66	56.40	65.50	60.82	59.06	60.29	76.70	59.72	16.98	60.99	58.46	59.70				

Table A.3: Level 2 - requirements category

Classification Model	CV1		CV2		CV3		CV4		CV5		CV		Training Testing Overfitting		Validation	
	accuracy [%]	average accuracy [%]	recall [%]	recall [%]	Precision [%]	Recall [%]	F1									
BernoulliNB(alpha=2, binarize=0, fit_prior=True, class_prior=None)	44.19	42.86	45.00	42.50	45.00	45.00	43.91	53.74	42.00	11.74	69.64	75.00	72.22			
ExtraTreesClassifier(max_features=100, class_weight='balanced', min_impurity_split=0.7, random_state=42)	69.77	63.29	62.50	65.00	57.50	63.81	72.11	52.00	20.11	100.00	75.00	85.71				
KNeighborsClassifier(n_neighbors=40)	65.12	71.43	60.00	62.50	72.50	68.31	70.75	66.00	4.75	75.00	75.00	75.00				
LinearSVC(C=0.05, class_weight='balanced', dual=True, fit_intercept=True, intercept_scaling=1, loss='hinge', multi_class='ovr', penalty='l2', tol=1e-05)	72.09	78.57	72.50	77.50	80.00	76.13	91.84	78.00	13.84	100.00	87.50	93.33				
LogisticRegression(n_jobs=1, multi_class='multinomial', class_weight='balanced', solver='lbfgs', C=0.01, penalty='l2')	83.72	83.33	80.00	72.50	80.00	79.91	93.88	72.00	21.88	100.00	87.50	93.33				
NearestCentroid()	79.07	80.95	87.50	80.00	80.00	81.50	94.55	74.00	20.55	75.00	75.00	75.00				
RandomForestClassifier(class_weight='balanced', min_impurity_split=0.7, max_depth=7, random_state=42)	60.47	73.81	62.50	60.00	52.50	61.85	78.23	46.00	32.23	66.67	50.00	57.14				
SGDClassifier(alpha=5, loss='log', penalty='l2', class_weight='balanced', n_iter_no_change=3, early_stopping=True, random_state=42)	41.86	33.33	42.50	42.50	5.00	33.04	43.54	34.00	9.54	87.50	37.50	52.50				

Table A.4: Level 2 - system category

Classification Model	CV1		CV2		CV3		CV4		CV5		CV		Training Testing Overfitting		Validation	
	accuracy [%]	average accuracy [%]	recall [%]	recall [%]	Precision [%]	Recall [%]	F1									
BernoulliNB(alpha=2, binarize=0, fit_prior=True, class_prior=None)	72.31	69.23	79.37	87.30	74.19	76.48	86.24	79.45	6.79	59.88	66.67	63.09				
ExtraTreesClassifier(max_features=50, class_weight='balanced', min_impurity_split=0.5, random_state=42)	67.69	76.92	73.02	80.95	79.03	75.52	90.37	80.82	9.55	75.44	74.07	74.75				
KNeighborsClassifier(n_neighbors=25)	76.90	76.90	87.30	80.95	72.58	78.94	79.36	73.97	5.39	81.14	70.37	75.37				
LinearSVC(C=0.05, class_weight='balanced', dual=True, fit_intercept=True, intercept_scaling=1, loss='hinge', multi_class='ovr', penalty='l2', tol=1e-05)	72.31	76.92	88.89	76.19	72.58	77.38	93.12	69.86	23.26	86.64	74.07	79.86				
LogisticRegression(n_jobs=1, multi_class='multinomial', class_weight='balanced', solver='lbfgs', C=0.01, penalty='l2')	78.46	66.15	88.89	77.78	79.03	78.06	91.74	79.45	12.29	83.30	81.48	82.38				
NearestCentroid()	83.08	78.46	92.06	85.70	82.26	84.30	93.12	86.30	6.82	85.19	85.19	85.19				
RandomForestClassifier(class_weight='balanced', min_impurity_split=0, max_depth=4, random_state=42)	69.23	61.54	63.49	71.43	74.19	67.98	83.94	73.97	9.97	62.12	62.96	62.54				
SGDClassifier(alpha=0.5, loss='log', penalty='l2', class_weight='balanced', n_iter_no_change=3, early_stopping=True, random_state=42)							57.80	56.16	1.64	26.89	51.85	35.41				

Table A.5: Level 2 - development process category

Classification Model	CV1		CV2		CV3		CV4		CV5		CV		Training Testing		Overfitting		Validation	
	accuracy [%]	average accuracy [%]	recall [%]	recall [%]	recall [%]	recall [%]	Recall [%]	Recall [%]	F1									
BernoulliNB(alpha=1, binarize=0, fit_prior=True, class_prior=None)	78.26	81.81	81.81	81.81	81.81	81.81	81.81	81.81	81.81	81.11	87.80	82.14	5.66	100.00	100.00	100.00	100.00	100.00
ExtraTreesClassifier(max_features=50, class_weight='balanced', min_impurity_split=0.4, random_state=42)	82.61	86.36	81.81	86.36	81.81	83.79	97.86	89.29	8.57	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
KNeighborsClassifier(n_neighbors=10)	78.26	81.81	81.81	81.81	81.81	81.11	81.70	82.14	-0.44	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
LinearSVC(C=0.1, class_weight='balanced', dual=True, fit_intercept=True, intercept_scaling=1, loss='hinge', multi_class='ovr', penalty='l2', tol=1e-05)	91.30	68.18	72.72	81.81	77.27	78.57	91.46	78.57	12.89	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
LogisticRegression(n_jobs=1, class_weight='balanced', solver='lbfgs', C=0.1, penalty='l2')	82.61	81.81	86.36	95.45	95.45	88.34	97.56	85.71	11.85	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
NearestCentroid()	82.61	81.81	86.36	95.45	95.45	88.34	97.56	89.29	8.27	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
RandomForestClassifier(class_weight='balanced', min_impurity_split=0, max_depth=5, random_state=42)	86.96	77.27	86.36	81.81	90.91	84.66	93.90	82.14	11.76	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
SGDClassifier(alpha=0.5, loss='log', penalty='l2', class_weight='balanced', n_iter_no_change=3, early_stopping=True, random_state=42)	86.96	68.18	77.27	95.45	95.45	84.66	97.56	89.29	8.27	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00

Table A.6: Level 3 - structure category

Classification Model	CV1		CV2		CV3		CV4		CV5		CV		Training Testing		Overfitting		Validation	
	accuracy [%]	average accuracy [%]	recall [%]	recall [%]	recall [%]	recall [%]	Precision [%]	Recall [%]	F1									
BernoulliNB(alpha=3, binarize=0, fit_prior=True, class_prior=None)	77.78	77.78	77.78	82.86	79.41	79.12	82.92	73.81	9.11	56.25	75.00	64.29						
ExtraTreesClassifier(max_features=50, class_weight='balanced', min_impurity_split=0.4, random_state=42)	91.67	77.78	86.11	88.57	85.29	85.88	96.75	73.81	22.94	83.33	83.33	83.33						
KNeighborsClassifier(n_neighbors=20)	86.11	80.56	83.33	88.57	85.29	84.77	85.37	73.81	11.56	86.36	83.33	84.82						
LinearSVC(C=0.001, class_weight='balanced', loss='hinge', penalty='l2', random_state=42)	58.33	50.00	55.56	45.71	55.56	53.10	79.67	73.81	5.86	84.38	58.33	68.98						
LogisticRegression(n_jobs=1, class_weight='balanced', C=0.01, penalty='l2')	66.67	72.22	61.11	54.29	67.65	64.39	91.06	73.81	17.25	90.00	83.33	86.54						
NearestCentroid()	91.67	88.89	91.67	85.71	79.41	87.47	98.37	73.81	24.56	83.33	83.33	83.33						
RandomForestClassifier(class_weight='balanced', min_impurity_split=0, max_depth=5, random_state=42)	77.78	77.78	77.78	85.71	73.53	78.52	88.62	73.81	14.81	86.36	83.33	84.82						
SGDClassifier(alpha=0.5, loss='log', penalty='l2', class_weight='balanced', n_iter_no_change=3, early_stopping=True, random_state=42)	83.33	77.78	77.78	85.71	70.59	79.04	88.62	73.81	14.81	56.25	75.00	64.29						

Table A.7: Level 3 - UI design category

Classification Model	CV1		CV2		CV3		CV4		CV5		CV		Training Testing Overfitting		Validation	
	accuracy [%]	average accuracy [%]	recall [%]	recall [%]	Precision [%]	Recall [%]	F1									
BernoulliNB(alpha=3, binarize=0, fit_prior=True, class_prior=None)	80.00	86.67	73.33	80.00	85.71	81.14	81.63	70.59	11.04	76.56	87.50	81.67				
ExtraTreesClassifier(max_features=50, class_weight='balanced', min_impurity_split=0.5, random_state=42)	20.00	20.00	20.00	20.00	78.57	31.71	81.63	70.59	11.04	76.56	87.50	81.67				
KNeighborsClassifier(n_neighbors=5)	80.00	86.67	73.33	80.00	85.71	81.14	81.63	70.59	11.04	76.56	87.50	81.67				
LinearSVC(C=0.0001, class_weight='balanced', loss='squared_hinge', penalty='l2', random_state=42)	53.33	60.00	60.00	86.67	42.86	60.57	87.76	70.59	17.17	89.29	25.00	39.06				
LogisticRegression(n_jobs=1, class_weight='balanced', C=0.001, penalty='l2')	66.67	66.67	53.33	93.33	42.86	64.57	95.92	70.59	25.33	89.29	25.00	39.06				
NearestCentroid()	80.00	86.67	73.33	80.00	78.57	79.71	100.00	70.59	29.41	75.00	75.00	75.00				
RandomForestClassifier(class_weight='balanced', min_impurity_split=1, max_depth=2, random_state=42)	80.00	73.33	66.67	80.00	71.43	74.29	91.84	70.59	21.25	76.56	87.50	81.67				
SGDClassifier(alpha=10, loss='log', penalty='l2', class_weight='balanced', n_iter_no_change=3, early_stopping=True, random_state=42)	80.00	86.67	73.33	80.00	21.43	68.29	93.88	70.59	23.29	89.58	37.50	52.87				

Table A.8: Level 3 - quality assurance category

Classification Model	CV					Training Testing Overfitting		Validation				
	CV1 accuracy [%]	CV2 accuracy [%]	CV3 accuracy [%]	CV4 accuracy [%]	CV5 accuracy [%]	average accuracy [%]	recall [%]	recall [%]	Precision [%]	Recall [%]	F1	
BernoulliNB(alpha=2, binarize=0, fit_prior=True, class_prior=None)	73.68	73.68	72.22	76.47	76.47	74.51	77.27	73.91	3.36	100.00	100.00	100.00
ExtraTreesClassifier(max_features=50, class_weight='balanced', min_impurity_split=0.5, random_state=42)	84.21	78.95	77.78	82.35	94.12	83.48	83.33	73.91	9.42	100.00	100.00	100.00
KNeighborsClassifier(n_neighbors=20)	74.51	73.68	72.22	76.47	76.47	74.51	74.24	73.91	0.33	100.00	100.00	100.00
LinearSVC(C=0.1, class_weight='balanced', loss='hinge', penalty='l2', random_state=42)	73.68	73.68	72.22	76.47	76.47	74.51	74.24	73.91	0.33	100.00	100.00	100.00
LogisticRegression(n_jobs=1, multi_class='multinomial', class_weight='balanced', solver='lbfgs', C=0.0001, penalty='l2')	57.89	63.16	55.56	58.82	70.59	61.20	87.87	73.91	13.96	100.00	100.00	100.00
NearestCentroid()	68.42	73.68	83.33	82.35	82.35	78.03	100.00	73.91	26.09	100.00	100.00	100.00
RandomForestClassifier(class_weight='balanced',73.68 min_impurity_split=0, max_depth=2, random_state=42)	73.68	89.47	66.67	70.59	82.35	76.55	89.39	73.90	15.49	100.00	100.00	100.00
SGDClassifier(alpha=0.1, loss='log', penalty='l2', class_weight='balanced', n_iter_no_change=3, early_stopping=True, random_state=42)	73.68	73.68	72.22	76.47	76.47	74.50	89.39	73.13	16.26	100.00	100.00	100.00

Table A.9: Level 4 - structure implementation category

Classification Model	CV1		CV2		CV3		CV4		CV5		CV		Training Testing Overfitting		Validation	
	accuracy [%]	average accuracy [%]	recall [%]	recall [%]	Precision [%]	Recall [%]	F1									
BernoulliNB(alpha=3, binarize=0, fit_prior=True, class_prior=None)	67.86	67.86	71.43	67.86	67.86	67.86	68.57	81.44	75.75	5.69				36.00	60.00	45.00
ExtraTreesClassifier(max_features=50, class_weight='balanced', min_impurity_split=0.4, random_state=42)	92.86	85.71	96.43	96.43	89.29	92.14	92.14	95.88	75.75	20.13				80.00	80.00	80.00
KNeighborsClassifier(n_neighbors=25)	92.86	92.86	92.86	82.14	82.14	88.57		87.63	75.75	11.88				57.50	60.00	58.72
LinearSVC(C=0.0001, class_weight='balanced', loss='squared_hinge', penalty='l2', random_state=42)	96.43	75.00	85.71	100.00	78.57	87.14		97.94	75.75	22.19				82.85	70.00	75.88
LogisticRegression(n_jobs=1, class_weight='balanced', C=1, penalty='l1')	82.14	75.00	82.14	89.29	78.57	81.43		83.51	75.75	7.76				69.52	70.00	69.76
NearestCentroid()	100.00	100.00	92.86	96.43	100.00	82.14		100.00	75.75	24.25				86.67	80.00	83.20
RandomForestClassifier(class_weight='balanced', min_impurity_split=0, max_depth=2, random_state=42)	78.57	78.57	85.71	75.00	89.29	81.43		90.72	75.75	14.97				91.99	90.00	90.98
SGDClassifier(alpha=0.1, loss='log', penalty='l2', n_iter_no_change=3, early_stopping=True, random_state=42)	67.86	67.86	67.86	67.86	67.86	67.86		81.44	75.75	5.69				36.00	60.00	45.00

Table A.10: Level 4 - behaviour implementation category

Classification Model	CV1		CV2		CV3		CV4		CV5		CV		Training Testing Overfitting		Validation	
	accuracy [%]	average accuracy [%]	recall [%]	recall [%]	Precision [%]	Recall [%]	F1									
BernoulliNB(alpha=10, binarize=0, fit_prior=True, class_prior=None)	64.29	50.00	58.33	50.00	72.72	59.07	93.02	53.33	39.69	100.00	33.33	50.00				
ExtraTreesClassifier(max_features=20, class_weight='balanced', min_impurity_split=0.6, random_state=42)	50.00	41.67	50.00	41.67	63.63	49.39	74.42	46.67	27.75	100.00	100.00	100.00				
KNeighborsClassifier(n_neighbors=20)	71.43	41.67	75.00	33.33	63.63	57.01	65.11	53.33	11.78	100.00	100.00	100.00				
LinearSVC(C=0.0001, class_weight='balanced', loss='squared_hinge', penalty='l2', random_state=42)	71.43	58.33	58.33	50.00	63.63	60.35	97.67	46.67	51.00	100.00	33.33	50.00				
LogisticRegression(n_jobs=1, class_weight='balanced', C=1, penalty='l1')	42.86	50.00	50.00	50.00	54.54	49.48	51.16	46.67	4.49	100.00	100.00	100.00				
NearestCentroid()	78.57	75.00	75.00	58.33	63.63	70.11	100.00	46.67	53.33	100.00	100.00	100.00				
RandomForestClassifier(class_weight='balanced', min_impurity_split=0, max_depth=2, random_state=42)	35.71	50.00	75.00	58.33	54.54	54.72	81.40	53.33	28.07	0.00	0.00	0.00				
SGDClassifier(alpha=0.1, loss='log', penalty='l2', n_iter_no_change=3, early_stopping=True, random_state=42)	-	-	-	-	-	-	-	-	-	-	-	-				

B

Appendix B

The following figure shows the results from the feature, functional requirement and use case clustering, using the K-means algorithm. Datapoints that were clustered differently when compared to manual labelling, were marked with red borders.

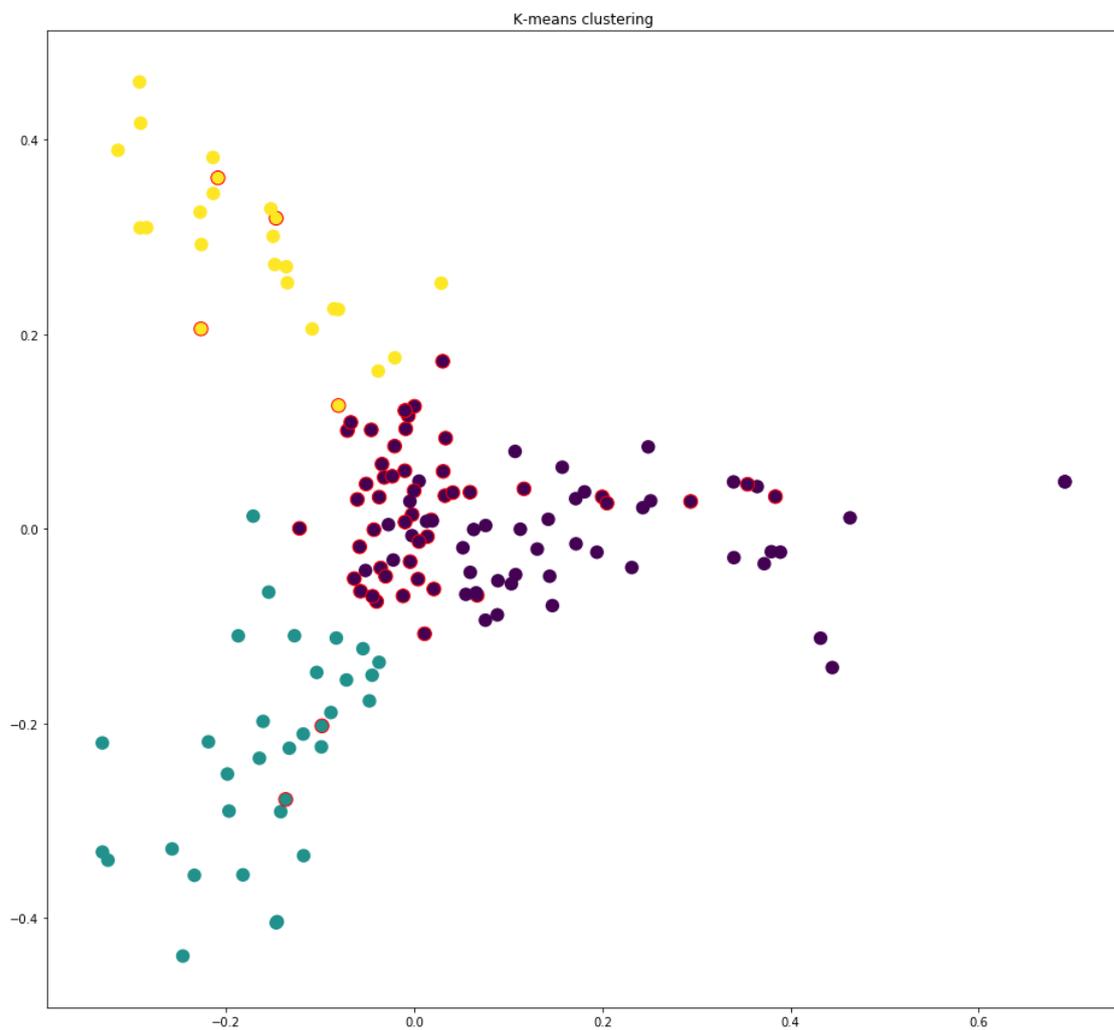


Figure B.1: Clustering of feature, functional requirement and use case sentences, using the K-means model

C

Appendix C

This appendix shows test that were made to check the normality and variance of the usability task times. Figure C.1 shows the normality results for the documentation task times and figure C.2 shows the normality results for the SDS task times. Figure C.3 then shows that results from the F-test, which was used to check whether the variance of the two samples was equal.

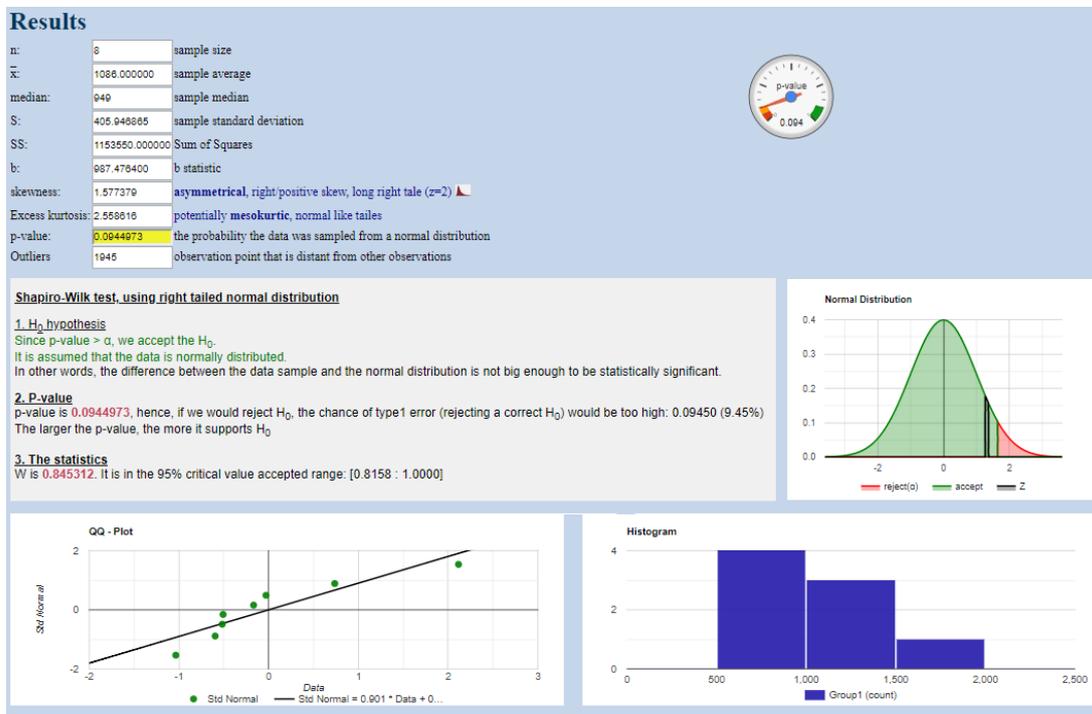


Figure C.1: Shapiro-Wilk and QQ-plot for documentation task times



Figure C.2: Shapiro-Wilk and QQ-plot for SDS task times



Figure C.3: F-test that checks for equal variance in the samples

D

Appendix D

This appendix lists all elements of the usability tests that were carried out. The first section shows the questionnaire that all participants answered before attending their usability test session. The second section then shows all elements of the usability testing session.

D.1 Personal experience questionnaire

Question 1 What is your highest level of education and in what field?

Question 2 Please indicate how many years of experience you have as a software developer?

Question 3 Please indicate your level of experience within web development?

- None
- Low
- Average
- High

Question 4 Please indicate which ones of the items below best describes your experience with Requirement analysis.

- Never worked with it
- Know about it but have never used it to perform development tasks
- I have used requirement analysis documents to perform development tasks
- I have conducted a requirement analysis

Question 5 Please indicate which ones of the items below best describes your experience with software architecture.

- Never worked with it
- Know about it but have never used it to perform development tasks
- I have used software architecture documents to perform development tasks
- I have designed software architecture

D.2 Usability test session

D.2.1 Introduction

Welcome to this system usability session and thank you again for participating in these usability tests.

The purpose of this session is to evaluate the system we created, which is called the Software Documentation Supporter (SDS), and is supposed to help users get desired information from long software documentation texts, without having to perform the tedious task of reading the whole documentation.

This session should take about 60 minutes and is divided into three parts.

1. A 10 minute session where you watch a video tutorial that should teach you the basics of the system and then play around in the system for a few minutes to get a better feeling of how it works.
2. A 40 minute session where you solve 4 tasks using the SDS and then solve 4 similar tasks using only the written documentation (about 5 minutes per task).
3. Answer a short user experience questionnaire.

What we are mostly interested in today is to observe how you use the tool to solve simple tasks about the system discussed in the documentation text.

Keep in mind that we don't expect you to gain an in-depth understanding of the system discussed in the documentation, and there are no right or wrong answers to the questions we are asking. Answer the questions to the best of your abilities, without focusing too much on finding the perfect answers.

Also keep in mind that in the SDS we are utilizing automatic methods such as machine learning. The accuracy of these results are never 100%, so use it to help you, but don't trust it blindly and let it replace your own thinking.

While solving the tasks, we can answer questions about the SDS or if you need clarification regarding the questions asked. If you are having troubles, we can also give you tips on how to use the system or where you might find the answers.

Finally, one note about the documentation text. There are a few pages from the original documentation that are not included in the version you have, for example, the introduction chapter. These pages did not contain any information relevant for solving the tasks and hardly even any information relevant to the implementation of the system. The information on these pages were not used in the implementation of the SDS and were therefore also removed from the PDF file that you will review here today.

If something is unclear during the process, don't hesitate to ask.

D.2.2 Tasks to solve using documentation text

Solve the following four tasks using the Snowflake documentation

Use the following new user story to answer question 1 and 2

User story

Name: Sort by date

Description: As a customer I want to be able to sort products based on how new they are, so I can see first the products that are new.

Question 1

Looking at the features that exist in the system:

- What existing feature is most relevant/suitable to include the new “Sort by date” functionality?
- Explain briefly why you chose this feature.

Question 2

Considering the feature display products (alternative term: browse products):

- Are there any additions or changes to use cases, requirements and/or user stories related to this feature that you think should be made because of the new “Sort by date” functionality? If so, which are affected? Please elaborate.

Question 3

What functionalities can the user apply to his cart?

Can you find source code related to these functionalities?

Question 4

What programming languages are used in the system implementation?

D.2.3 Tasks to solve using the Software Documentation Supporter

Solve the following four tasks using the Software Documentation Supporter (SDS)

Use the following new user story to answer question 1 and 2

User story

Name: Shipping address same as billing address

Description: As a customer, I want to be able to save the information that my shipping address is the same as my billing address, so that I don't always have to fill it in twice when I am ordering products.

Question 1

Looking at the features that exist in the system:

- What existing feature is the most relevant/suitable to include the new “Shipping address same as billing address” functionality?
- Explain briefly why you chose this feature.

Question 2

Considering the feature user management (alternative terms: account management or manage account):

- Are there any additions or changes to use cases or requirements related to this feature that you think should be made because of the new “Shipping address same as billing address” functionality? If so, which are affected? Please elaborate.

Question 3

How are products sorted on the home page?

Can you find source code of the sorting implementation?

Question 4

What architecture patterns are used in the system?

D.2.4 User experience interview

Please answer the following questionnaire (System Usability Scale) according to the instructions given. Then fill in the blanks for the four questions below.

Please note that both positive and negative feedback on the system is valuable, so do your best to give your honest opinion.

System Usability Scale

Instructions: For each of the following statements, mark one box that best describes your reactions to the website today.

	Strongly disagree				Strongly agree
I think that I would like to use this website frequently.	<input type="checkbox"/>				
I found this website unnecessarily complex.	<input type="checkbox"/>				
I thought this website was easy to use.	<input type="checkbox"/>				
I think that I would need assistance to be able to use this website.	<input type="checkbox"/>				
I found the various functions in this website were well integrated.	<input type="checkbox"/>				
I found the various functions in this website were well integrated.	<input type="checkbox"/>				
I would imagine that most people would learn to use this website very quickly.	<input type="checkbox"/>				
I found this website very cumbersome/awkward to use.	<input type="checkbox"/>				
I felt very confident using this website.	<input type="checkbox"/>				
I needed to learn a lot of things before I could get going with this website.	<input type="checkbox"/>				

* This questionnaire is based on the System Usability Scale (SUS), which was developed by John Brooke while working at Digital Equipment Corporation. © Digital Equipment Corporation, 1986.

D. Appendix D

1. In comparison to using normal documentation, I like...
2. If I was to use this tool in practice, I would like these changes or additions...
3. When using the system, I felt ...
4. I additionally have the following comments about the website...

E

Appendix E

This appendix shows a summary of the usability testing participants' experience.

Participant ID	Highest level of Education	Years of experience in software engineering	Level of experience with web development	Level of experience in requirements analysis	Level of experience in software architecture
P1	MSc Software Engineering	0.5	average	I have used requirement analysis documents to perform development tasks. I have conducted a requirement analysis.	I have used software architecture documents to perform development tasks.
P2	PhD Computer Science	10	average	I have used requirement analysis documents to perform development tasks. I have conducted a requirement analysis.	I have used software architecture documents to perform development tasks.
P3	PhD Software Engineering	3	high	I have used requirement analysis documents to perform development tasks. I have conducted a requirement analysis.	I have designed software architecture.
P4	BSc Software Engineering Licentiate Software Engineering	5	low	I have conducted a requirement analysis.	I have designed software architecture.
P5	PhD Computer Science/Software Engineering	4	low	I have conducted a requirement analysis	I have designed software architecture.
P6		15	average	I have conducted a requirement analysis documents to perform development tasks. I have conducted a requirement analysis. I have used requirement analysis documents to perform development tasks.	I have designed software architecture.
P7	PhD Computer Science	8	high	I have used requirement analysis documents to perform development tasks. I have conducted a requirement analysis. Know about it but have never used it to perform development tasks.	I have designed software architecture.
P8	MSc Software Engineering	5	average	I have conducted a requirement analysis.	I have designed software architecture.
P9	Licentiate Computer Science	20	low	I have conducted a requirement analysis.	I have designed software architecture.

Figure E.1: Results from the personal experience questionnaire

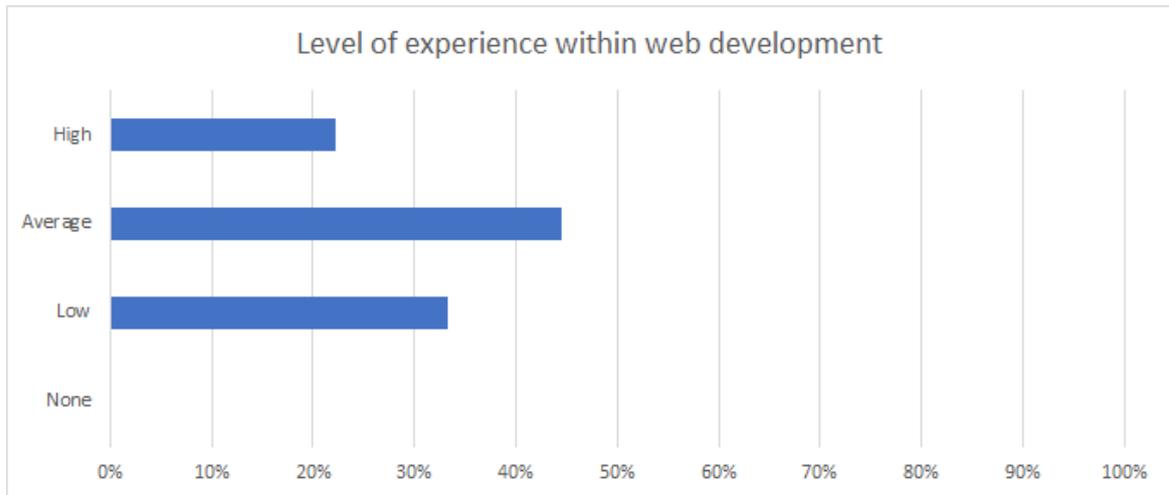


Figure E.2: A bar chart showing the participants' experience with web development

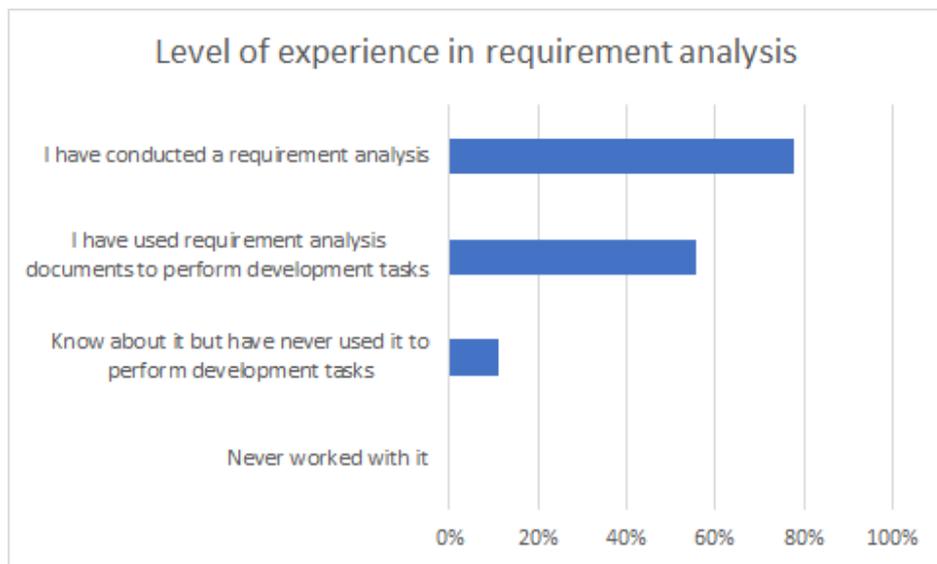


Figure E.3: A bar chart showing the participants' experience with requirement analysis

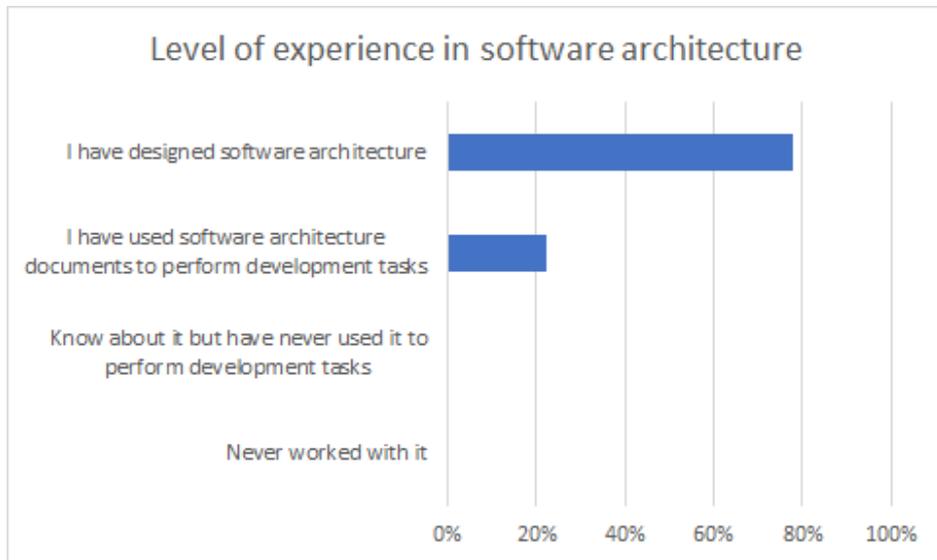


Figure E.4: A bar chart showing the participants' experience with software architecture

F

Appendix F

This appendix shows the results from the user experience interview. The first section shows results from the SUS questionnaire and the second section shows all comments received through the open-ended questions.

F.1 Results from SUS questionnaire

Question	P1	P2	P3	P4	P5	P6	P7	P8	P9	Average score per question
1	7.5	7.5	2.5	5	10	2.5	2.5	7.5	5	55.56
2	7.5	10	7.5	7.5	7.5	10	2.5	7.5	5	72.22
3	5	7.5	7.5	7.5	7.5	7.5	7.5	5	5	66.67
4	10	10	5	7.5	7.5	10	2.5	7.5	5	72.22
5	5	10	10	7.5	5	7.5	7.5	5	2.5	66.67
6	5	10	2.5	7.5	7.5	10	7.5	10	5	72.22
7	10	7.5	7.5	7.5	7.5	7.5	5	7.5	7.5	75.00
8	7.5	10	7.5	7.5	7.5	7.5	5	10	5	75.00
9	5	10	2.5	5	5	2.5	2.5	7.5	2.5	47.22
10	10	10	7.5	7.5	7.5	5	7.5	10	7.5	80.56
Total score per participant:	72.5	92.5	60	70	72.5	70	50	77.5	50	68.33

Figure F.1: Results from the System Usability Scale questionnaire

F.2 Results from open-ended questions

Participant ID	Relates to	Type	Comment
P1	Traceability / Navigation / Overview	Like	good with having the real documentation alongside.
P1	Navigation	Like	easier to navigate to different sections or chapters of the documentations
P1	Learnability	Like	support in finding answers to my questions
P2	Learnability	Like	The system gave me confidence that I had found all instances of the description of a particular functionality (or similar), whereas with the pdf, it was not clear if I had found all instances of what I was looking for.
P2	Visualisation / Navigation	Like	The system also seemed to highlight important aspects of the documentation.
P3	Navigation / Learnability	Like	That you can query the documentation. It's not a basic search facility, but your search questions carry a specific semantics which makes the search easier.
P4	Navigation	Like	Not having to scroll back and forth and constantly going back to the contents section to find what I'm looking for.
P4	Navigation / Learnability	Like	It seems like a very nice way to find information quickly from a large and extensive documentation.
P5	Navigation / Overview	Like	The summaries. Features or requirements where summarized/categorized. That's a help I think even though it might not be 100% correct.
P6	Visualisation / navigation	Like	The ability to combine filtering and searching
P6	Visualisation / navigation	Like	The ability to easily turn filters on and off.
P7	Visualisation	Like	The graph structure and categorization of sentences.
P7	Visualisation	Like	Determine relationships between search keywords and some sentences.
P7	Navigation / Overview	Like	Capture general information about the system (e.g. technologies and patterns)
P8	Visualisation / navigation	Like	things
P8	Visualisation / navigation	Like	That the display is graphical which makes it easy to reason.

Figure F.2: A list of comments about what participants liked about the SDS

Participant ID	Relates to	Type	Comment
P1	Incompleteness	Feel	I not would trust the results completely
P1	Interested	Feel	Exciting to see new ways for attempting to solve the issues of documentation
P2	Confident	Feel	more confident in answering the questions.
P2	Supported	Feel	that it provided me with a more natural way of navigating documentation (i.e., search-based vs. Sequential reading of a pdf document)
P3	Incompleteness	Feel	Afraid. I felt as if there is a certain risk to miss something of importance. So while the questions help in querying the document, they might also very well miss important content.
P4	Supported	Feel	Satisfied over being able to search for things in images instead of in text.
P4	Confused	Feel	A bit confused on which question to use when.
P5	Supported	Feel	Supported in my quest for answers
P5	Incompleteness	Feel	worried about the completeness of the results
P6	Confused	Feel	Often disoriented by the number of results I got. I found scanning the fragments without context (in order to decide which one to click on, to get more context) very difficult. This may be a chicken-and-egg problem 😊
P7	Interested	Feel	Quite a new experience, which needs some learning on how to use the system
P7	Interested	Feel	It was interesting to explore the document through a graph.
P8	Supported	Feel	I felt like I did not need to read the entire documentation.

Figure F.3: A list of comments regarding how participants felt when using the SDS

Participant ID	Relates to	Type	Comment
P1	Feature suggestion	Improvement suggestion	Machine learning algorithm that summarizes the information for me
P1	Improve result design	Improvement suggestion	More detailed clustering of items in the graph
P1	Improve result design	Improvement suggestion	A way to visually link items on the graph to the text
P1	ML improvements	Improvement suggestion	Improving the machine learning algorithm
P2	Improve result design	Improvement suggestion	Representation of hierarchical categories?
P2	Improve result design	Improvement suggestion	Highlight where in the documentation the search results are
P2	Improve UI design	Improvement suggestion	Hierarchy of original documentation content
P2	Improve UI design	Improvement suggestion	Make graph less prominent
P2	Searchable drop-down/results	Improvement suggestion	Search and/or filtering functionality in the different parts of the website.
P2	Searchable drop-down/results	Improvement suggestion	Auto-complete for drop down menus
P4	Other	Improvement suggestion	A more clear distinction of when to use which kind of question. Probably this comes naturally with more experience though.
P5	Improve result design	Improvement suggestion	I would like some indication of tool confidence. Like, „we think this is a pattern 99% confidence.“
P5	Searchable drop-down/results	Improvement suggestion	The list of features/requirements can be long. Might be good to have a searchable drop down list instead
P6	Improve result design	Improvement suggestion	Minor UX issues such as an indication of how many results there are, and which one I'm currently looking at (otherwise it's easy to lose track at which one of all the yellow marks I am).
P6	Improve UI design	Improvement suggestion	Clearing the results window when a new question is selected.
P6	ML improvements	Improvement suggestion	Initial results are often overwhelming due to large number of displayed fragments. Would be nice if initial results could already be more focused.
P7	Improve UI design	Improvement suggestion	Simplify UI - Refine/categorize the list of use-cases.
P7	Improve result design	Improvement suggestion	Simplify UI - Guide users to suitable question
P7	Improve result design	Improvement suggestion	Simplify UI - Simplify the relationships in graph
P8	Feature suggestion	Improvement suggestion	Have the possibility to click on the highlighted text and go back to the diagram.
P8	Improve UI design	Improvement suggestion	Make the scrolling more natural, it is difficult to scroll in the image now
P9	Improve UI design	Improvement suggestion	Change the de-selection of the button. I found it confusing/not clear as it is now
P9	Improve UI design	Improvement suggestion	Add some more colors to the screens and make it possible to change the size of the letter (quite difficult to read the text in the cascade picture)

Figure F.4: A list of improvement suggestions from the participants

Participant ID	Relates to	Type	Comment
P1		Other	Covering weaknesses of the machine learning algorithm by providing actual documentation beside was interesting
P3		Other	I think the website has potential given a couple of constraints: For querying something you are completely unfamiliar with (like the example system), I fear that it's way too easy to miss important content, or misunderstand how the system works. The opposite situation, in which you are rather familiar with the system, is equally bad, as you are probably familiar enough with the spec to locate the „relevant“ parts yourself pretty quickly (and quicker than using the website). Instead, I believe the website would be helpful if the system under development is (1) large enough that makes the spec hard to overview, (2) if it is developed by many teams, so that each team has their own area of responsibility, and (3) the specification has a similar structure for each component/area. That way, the spec is complex enough so that you need help, and at the same time the structure is familiar, so that you can be confident in not having missed anything.
P8		Other	Record the screen so that you know how users used the tool.
P9		Other	I guess it would take some time to get familiar with the software, but after a while I think it would be a useful tool. However, it is not clear, WHO the user is and I got the impression that the user needs to be quite familiar with both the agile way of thinking and also technical documentation in general.
P9		Other	The zooming needs some extra UX-work
P9		Other	The questions in the questionnaire did me a bit confused...otherwise, and after a short learning period I think I could use the tool successfully

Figure F.5: A list of other comments received from the participants

G

Appendix G

This appendix lists all pointers given in the usability tests.

Participant ID	Task	Pointer
P4	SDS – task 1	Question rephrased to clarify its meaning.
P4	SDS – task 3	A pointer given that the question was referring to implementation of the UI
P5	Documentation – task 1	Clarification of the question – you should think about the features of the system and its functionalities and find where it fits best.
P5	Documentation – task 2	Explained the feature terms used in the documentation, that display product, list product and browse products are all synonyms Gave him a tip on where in the documentation he might find the answer
P5	SDS – task 1	Explained to him the difference between features and functional requirement, as I define it in this research.
P6	SDS – task 3	Explained to him that the sentence he was looking for did not necessarily have to contain the words „by default“. He had found the correct answer but was unsure about it because of how the question was phrased.
P7	Documentation – task 2	Explained to him what kind of answer we expect, so just a question clarification
P7	SDS – task 3	Gave him a tip on what question to ask the system. It turned out he had misunderstood the question, he had already found the answer we were looking for, but assumed that we were looking for implementation details.
P9	Documentation – task 1	Question rephrased to clarify its meaning.
P9	Documentation – task 2	Question rephrased to clarify its meaning. Participant asked for confirmation about the answer she had in mind – this confirmation was given.
P9	Documentation – task 3	Participant asked for confirmation about the answer she had in mind – this confirmation was given.
P9	SDS – task 1	Information given about how the filter buttons in the system work. Question rephrased to clarify its meaning.
P9	SDS – task 3	Participant reminded that the sentences in the results in the system are clickable.

Figure G.1: Pointers given during usability testing