Eileen Dickson

Utilization of Deep Learning for the Automatic Classification of Software Aspects

Master's Thesis

Summer Semester 2022
Supervisors: Prof. Dr. Barbara Paech, Michael Anders
Institute for Computer Science
Ruprecht-Karls-University Heidelberg

July 18, 2022

Abstract

Context and Motivation: Users give natural language feedback for software in online forums, which can be used by developers to create requirements. In order to categorize user feedback into categories which are relevant for requirements engineering, feedback can be classified into different categories. One such approach is the classification into different software aspects. The TORE framework provides a basis for such a classification. The classification can be done manually by creating codes containing TORE categories for tokens, which can be done by one or multiple raters on the same documents. In order to limit mistakes stemming from the raters, the classifications can be compared by creating inter-rater agreements. This comparison can be supported by tools, which can simplify the process of resolving disagreements. However, manual classification is cost- and time-intensive. Therefore an algorithm is needed for the automatic classification of natural language user feedback into TORE-categories.

Contributions: This thesis contains three major contributions. Firstly, a tool support is provided for the comparison of an arbitrary number of annotations of the same dataset. Disagreements can be resolved by accepting or declining codes assigned to words. Additionally, agreement overviews as well as statistics are provided. Secondly, a literature review is conducted on automatic deep-learning-based classification of natural language user feedback from online forums. The review shows that many approaches use binary classification as a basis, although some use multi-class classification. Most of them use some kind of pre-processing, Word2Vec for word embeddings, and a neural network for the classification. Lastly, a classifier is implemented, based on a Bi-LSTM model for the automatic classification of natural language texts into TORE-categories. The classifier is trained on a manual annotation of forum data, and subsequently evaluated.

Results: A tool for the comparison of annotations is developed and implemented in Feed.UVL, and the quality is assured through extensive system tests and an informal usability test. The tool reuses existing views and functionalities, and is user-friendly in design, due to the use of multiple tool-tips and standardized icons. The underlying microservice architecture simplifies possible future extension of the algorithm. The automatic classification tool, based on a deep learning algorithm, is also implemented into Feed.UVL, and reuses the existing views and functionalities for the classification results. The *accuracy* achieved over all classes is about 52.74%, with a *recall* of 53.47%, *precision* of 54.59% and *F1-score* of 53.78%, which is mainly due to the small amount of training data, as well as a poorly chosen test dataset.

Zusammenfassung

Kontext und Motivation: NutzerInnen geben in Online-Foren natürlichsprachliches Feedback für Software, das von EntwicklerInnen verwendet werden kann, um Anforderungen zu erstellen. Einen Ansatz das Nutzerfeedback in Kategorien einzuordnen, die für die Anforderungsentwicklung relevant sind, bietet die Klassifikation in verschiedene Softwareaspekte. Hierfür bietet das TORE-Framework eine Grundlage. Die Klassifikation kann manuell erfolgen, indem Kodierungen erstellt werden, die TORE-Kategorien für Textstellen enthalten, welche von mehreren KodiererInnen an denselben Dokumenten vorgenommen werden können. Um Fehler der KodiererInnen einzuschränken, können die Klassifikationen durch das Erstellen von Interrater Agreements verglichen werden. Dieser Vergleich kann durch Tools unterstützt werden, die den Prozess der Lösung von Uneinigkeiten vereinfachen können. Die manuelle Klassifikation ist jedoch kosten- und zeitintensiv. Daher wird ein Algorithmus zur automatischen Klassifikation von natürlichsprachlichem Nutzerfeedback in TORE-Kategorien benötigt.

Beiträge: Diese Arbeit enthält drei Hauptbeiträge. Erstens wird eine Werkzeugunterstützung für den Vergleich einer beliebigen Anzahl von Annotationen desselben Datensatzes bereitgestellt. Uneinigkeiten können gelöst werden, indem den Wörtern zugeordnete Codes akzeptiert oder abgelehnt werden. Zusätzlich werden Übersichten sowie Statistiken bereitgestellt. Zweitens wird eine Literaturrecherche zur automatischen Deep-Learning-basierten Klassifikation von Nutzerfeedback in natürlicher Sprache aus Online-Foren durchgeführt. Die Übersicht zeigt, dass viele Ansätze eine binäre Klassifikation als Grundlage verwenden, und dass die meisten eine Art des Pre-Processing, Word2Vec für Word Embeddings und ein neuronales Netzwerk für die Klassifikation benutzen. Schließlich wird ein *Classifier* basierend auf einem Bi-LSTM-Modell zur automatischen Klassifikation von natürlichsprachlichen Texten in TORE-Kategorien implementiert. Der *Classifier* wird auf einer manuellen Annotation von Forumsdaten trainiert und anschließend ausgewertet.

Resultate: Ein Tool zum Vergleich von Annotationen ist in Feed.UVL entwickelt und implementiert, die Qualität wird durch umfangreiche Systemtests und einem informellen Nutzungstest sichergestellt. Das Tool verwendet vorhandene Sichten und Funktionalitäten und ist nutzerfreundlich gestaltet. Die zugrunde liegende Mikroservice-Architektur vereinfacht zukünftige Erweiterungen des Algorithmus. Das automatische Klassifizierungstool, das auf einem Deep-Learning Algorithmus basiert, ist ebenfalls in Feed.UVL implementiert und verwendet die vorhandenen Sichten und Funktionalitäten für die Klassifizierungsergebnisse. Die erreichte *Accuracy* über alle Klassen liegt bei etwa 52,74%, mit einem *Recall* von 53,47%, einer *Precision* von 54,59% und einem *F1-Score* von 53,78%, was hauptsächlich auf die geringe Menge an Trainingsdaten sowie einen schlecht gewählten Testdatensatz zurückzuführen ist.

Ich versichere, dass ich diese Master-Arbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe und die Grundsätze und Empfehlungen "Verantwortung in der Wissenschaft" der Universität Heidelberg beachtet wurden.

Abgabedatum: July 18, 2022

Contents

1 Introduction

This chapter serves as an introduction for this thesis. In section 1.1 the motivations for this paper are outlined, the goals are summarized in section 1.2, and in section 1.3 an overview of the rest of this thesis is provided.

1.1 Motivation

In the software engineering process, there are multiple ways to extract information from users, such as interviews, questionnaires, etc. While these extraction methods can be used in order to establish requirements, they can also be expensive, as they require a lot of effort: developing questions with which to best extract data from users; collecting data from users who take part in the data extraction methods; and lastly using the data to develop requirements.

Conveniently, users already give feedback in various forms, be it in the form of product reviews, in the app store, on *Twitter*, or in online forums. In the case of online forums, users can take part in discussions about software, either by asking and answering questions, or by stating their opinion about it. Those online forums form a large corpus about specific software, which have already been used as a basis to develop requirements (e.g. [10], [14], [8]), although they pose some challenges. Users mostly tend to formulate their feedback from the perspective of someone using the UI, and mostly have an external view of the software. As requirements engineers and developers tend to have an internal view of the software, meaning they tend to think about software components and processes within the software, it can be hard to grasp which parts of the software users are referring to. Classification of the feedback into categories which describe the mental models of the users could help developers understand the needs and requirements of users better.

The *Task-Oriented Requirements Engineering* (TORE) [17] framework, which was originally developed to help requirement engineers with integrating object-oriented development into the requirement engineering process, is used as a basis for the classification of user feedback. It consists of multiple layers of abstraction, which can be used by requirement engineers as a basis for decision processes and the definition of requirements. Feedback gathered from online forums can be annotated using the TORE framework, which means text passages of the user feedback can be assigned identifiers from the TORE framework. Additionally, as the user feedback from online forums is in natural language, it contains mistakes and incorrect grammar, making the analysis more difficult.

Furthermore, manual annotation of a dataset can be subjective, and depends on the annotator involved. In order to alleviate the subjectiveness, an agreement, which is a comparison of multiple annotations by different annotators, can be used for the classification process. Two or more annotations can be compared, and more objective identifiers for text passages can be assigned, which can be useful as basis for the training of an automatic classifier. The need to compare large annotations necessitates tool support for inter-rater agreement functionalities. Raters comparing annotations would require multiple days to weeks without proper tool support which can reduce the time needed dramatically. Lastly, manual annotation is cost- and time-intensive. A developer classifying 100 user feedback statements with the TORE framework would require about 15 hours, as experienced during the classification of a dataset for this thesis. The implementation of an automatic classifier could reduce the time to mere seconds.

1.2 Goals

In this thesis, there are two major goals:

1. **Extending the functionalities of the Feed.UVL[1] tool by adding the possibility to compare annotations to another, as well as calculating inter-rater agreements of two or more annotators.** Feed.UVL already provides the functionality to create annotations for natural language data, and the TORE framework can already be used as a basis to encode text-passages. With the addition of inter-rater functionalities, multiple annotations, and therefore the codes for text-passages, can be compared.

2. **Implementing a deep-learning-based classifier in Feed.UVL, in order to classify forum data automatically using the TORE framework.** The forum data consists of natural language documents, which contains information about the software. This feedback can be analysed automatically in order to find

[1]https://feed-uvl.ifi.uni-heidelberg.de/dashboard/login

out which components and aspects of the software users are most interested in, using the TORE framework.

1.3 Overview

First, in chapter 2, the fundamentals necessary to understand this thesis are introduced. A thorough literature review based on predefined research questions has been conducted, the results of which are documented in chapter 3. In the following chapter 4, the addition of the inter-rater functionalities are documented. The training dataset, implementation and evaluation of the deep-learning-based classifier are provided in chapter 5. Lastly, in chapter 6 the results of this thesis are summarized and possibilities for future work are outlined.

2 Fundamentals

The fundamental knowledge needed to understand this thesis is summarized in the following sections. In section 2.1 the basis for the annotation of datasets used in this paper, TORE, is explained. In section 2.2 annotations of datasets are introduced, manual and automatic annotations are differentiated, and inter-rater agreements are outlined. In section 2.3 the annotation process of a dataset using the TORE framework is explained using an example. In section 2.4, Feed.UVL is presented and the most important functionalities, the microservice architecture and the software used to implement it are outlined. Lastly, in section 2.5, the deep learning algorithm Bi-LSTM is introduced.

2.1 TORE

As mentioned in section 1.1, TORE stands for the Task-Oriented Requirements Engineering Framework, first introduced by Paech in [17]. It consists of decision points, which can be grouped into four abstraction layers, which can be observed in figure 2.1.

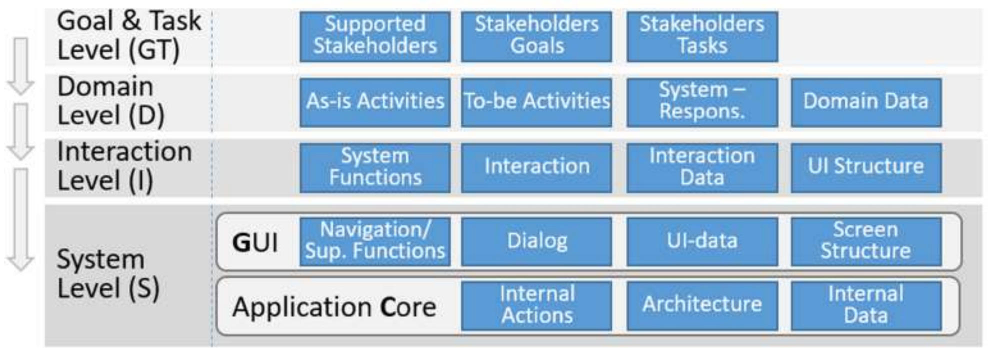


Fig. 2.1: The TORE model with four layers of abstraction. [1]

For the purposes of applying TORE to user feedback, a simplified model of the TORE framework is used, and the *Goal & Task Level* and *Domain Level* are summarized, resulting in a model with only three abstraction layers as shown in figure 2.2.

The first layer, the *Domain Level*, consists of decisions concerning the domain of the

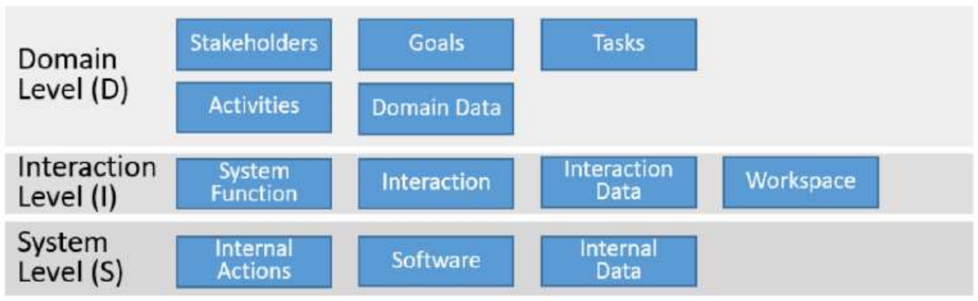


Fig. 2.2: The compressed TORE model with three layers of abstraction.

software. This includes decisions about the *Stakeholders*, as well as their *Goals*, *Tasks* and *Activities*. *Stakeholders* are people or entities that are using or influencing the software; stakeholders' *Tasks* are great responsibilities, which can be divided into many small activities. *Activities* consist of steps that realize the tasks, and can be performed without the software. Lastly, *Domain Data* is any data relevant to a task.

The second layer, the *Interaction Level*, consists of decisions regarding the *Interaction* of the system with the stakeholder. *System Functions* are functions provided by the system itself that manipulate data in some form. *Interactions* describe the interaction between users and the system, mostly through some form of interaction with the UI. *Interaction Data* is data which is relevant to a system function or an interaction. Lastly, the *Workspaces/UI-Structure* includes any grouping of system functions, interactions or UI-functionalities into *Workspaces*.

The last layer, the *System Level*, only includes software-internal data and functions, and is independent of the stakeholder. Decisions include *Internal Actions* and *Internal Data*, which consist of functions to realize the interactions internally, as well as data necessary for the internal processes. Decisions about *Software* contain all related software components for the realization of the software. A summary of the TORE-levels and the respective decisions can be observed in table 2.1.

Table 2.1: TORE decisions and their definitions. From Anders et al., [1].

Goal, Task, and Domain Level	
Stakeholders	Roles supported by or influencing the developed software
Stakeholders' Goals	Goals the software should fulfill
Stakeholders' Tasks	Responsibilities of the Stakeholder as part of larger processes in the domain
Activities	Steps in the Stakeholder Tasks
Domain Data	Data relevant to some activity
Interaction Level	
Interaction	The interaction between a user and the software Includes in addition the Dialog as a refinement of the Interactions into screen sequences
System Functions	Functions executed by the software that consume, manipulate or produce data Includes in addition the Navigation and Support Functions needed for the data related functions
Interaction Data	Data relevant for the System Functions Includes the UI-Data which refines the Interaction Data
UI-Structure (Workspace)	Grouping of Interaction Data and System Functions which are relevant for one Task into so-called Workspaces Includes Screen Structure as a refinement of the Workspaces
System Level	
Internal Actions	Steps needed to realize the Interaction Level
Architecture (Software)	Components of the software and their relationships
Internal Data	Data processed by Internal Actions

2.2 Annotation

In subsection 2.2.1, the terminology for the annotation process is defined. In subsection 2.2.2 manual annotations are presented, followed by an explanation of automatic annotations in subsection 2.2.3. Finally, in subsection 2.2.4 the concept of inter-rater agreements is provided.

2.2.1 Terminology

Firstly, to establish an understanding of annotations, there is some terminology to be defined.

Dataset. A dataset is a collection of documents that are related in some way. In the context of this thesis, the term dataset will be used to refer to a collection of documents with the same source, such as multiple posts from a single forum on a single topic.

Text Passage. A text passage is a cohesive unit of speech of variable length, which contains relevant information for an annotator. A text passage itself can consist of multiple sub-text passages.

Code. A code is an identifier that is assigned to a text passage.

Annotation/Encoding. An annotation is a version of an annotated/encoded dataset, which was established by an annotator. Consequently, a dataset can have multiple annotations, which can be distinguished by the number of code occurrences and the annotated text passages.

The terms annotation and encoding are synonymous, and are henceforth used interchangeably.

2.2.2 Manual Annotation

Manual annotation is a process in which annotations are created manually by a human being, usually an expert in the domain. This human expert, also called annotator, is given a dataset and a set of categories. The annotator is then asked to annotate the data, which in general means assigning codes to parts of the document. In this thesis, annotators are asked to assign codes to text passages. The resulting annotation can for example be used to train machine learning or deep learning algorithms.

Manual annotation is comparatively time consuming and costly. Additionally, annotating a dataset can take a long time, and when there is a lot of data, human annotators can become tired and less focused. Still, manual annotation tends to be relatively accurate.

2.2.3 Automatic Annotation

Automatic annotation is a process in which annotations are created automatically by a machine or algorithm. Algorithms used for automatic annotation are usually machine learning algorithms, with the dataset as the input of a pre-trained machine learning algorithm, and the annotation as the output.

Automatic annotation tends to be much less costly, as no human annotators have to be paid. Since algorithms can work with large amounts of data in a short time, automatic annotation requires less time overall. However, automatic annotation tends to be less accurate than manual annotation especially with natural language datasets, as semantic context and incorrect spelling and grammar can be problematic to understand for machine learning algorithms. Additionally, if the algorithm's model is not well trained, the accuracy suffers further.

2.2.4 Inter-Rater Agreement

The inter-rater agreement is based on the comparison of two or more annotations, and describes the degree to which annotations are identical. (Gisev et al., [7]) In order to find the degree to which annotators agree, their annotations have to be compared. More precisely, when annotators have encoded the same text passage with the same code, they have a higher degree of similarity. In this thesis, two kappa values are used to measure this similarity: The *Fleiss' Kappa* from [6] and the *Kappa of Brennan & Prediger* from [3].

Kappa values can be between 0 and 1, with higher values pointing to a higher degree of relatedness, and therefore a better inter-rater agreement. The kappa values can also be used as an indicator of how well machine learning algorithms may perform on a similar dataset, as high kappa values point to human annotators having no problems annotating datasets. A low kappa value could point to difficulties for human annotators in the annotation process of datasets.

2.3 TORE Encodings

In this section, the annotation process of a dataset with TORE categories is explained using an example sentence from the training dataset used for the classifier:

I'm road cycling and I would like to see the specific maps.

In the first step, the sentence is split into single words.

I / 'm / road / cycling / and / I / would / like / to / see / the / specific / maps / .

Afterwards, text passages that are relevant for the TORE classification are identified and connected.

I / 'm / road cycling / and / I / would / like / to / see / the / specific / maps / .

In the last step, the TORE categories are assigned to the relevant text passages.

I / 'm / road cycling / and / I / would / like / to / see / the / specific / maps / .
Activity Interaction Interaction Data

2.4 Feed.UVL

In section 2.4.1 the basic microservice architecture used in the Feed.UVL project is explained. Feed.UVL is the tool that is extended in this thesis, hence a short summary of the most relevant functionalities is provided in section 2.4.2. Lastly in section 2.4.3 the containerization application *Docker*, as well as *Jenkins*, an application supporting continuous integration, are introduced.

2.4.1 Mircoservice Architecture

Traditionally, applications have been managed as one service, which is known as a monolithic architecture. As there is only one service, development in monolithic applications can become more complex with time; the service can have conflicting dependencies and new features can have impact on existing features. Consequently, the scalability of single-service architectures is relatively low.

The mircoservice architecture encapsulates the idea to split a large application into smaller, independent services, which often only have a single concern. These microservices have no external dependencies, and can therefore be managed independently. The communication between microservices is through well defined APIs, so as long as the containers support those, they can easily be replaced. They are also highly scalable, as they are completely independent from other services. This independency can also improve speed, agility and consistency. A visualization of the monolithic and microservice architecture can be observed in figure 2.3.

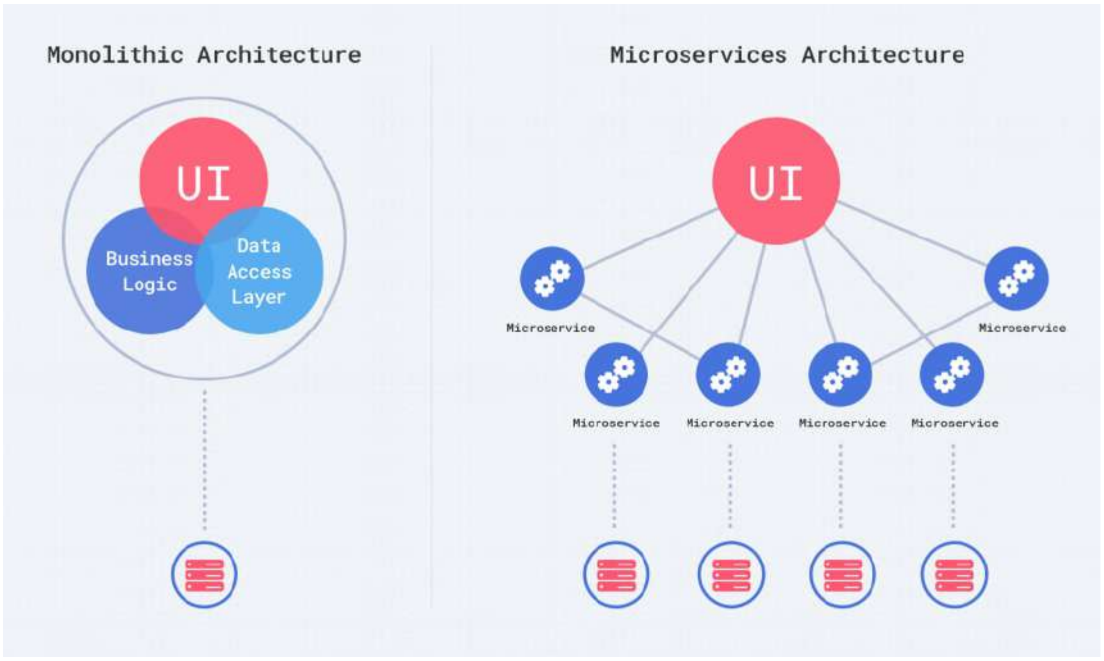


Fig. 2.3: The monolithic vs. microservice architecture[1].

There are several advantages of the microservice architecture, including the use of different programming languages, frameworks and databases for different concerns. A mircoservice for data analysis can be based on *Python*, whilst a frontend microservice can use *Javascript*. The microservices are also usually more resilient, so that if one service has an error, the rest of the application is not affected and can continue running.

The current mircoservice architecture used in the Feed.UVL project can be observed in figure 2.4. The project is divided into different layers by concern, with every service except the light blue ones already implemented. With this thesis, the light blue services will be added, including the agreement functionality to the application layer, and the TORE classification using a deep learning algorithm to the data analytics layer.

[1]https://dev.to/alex_barashkov/microservices-vs-monolith-architecture-4l1m, last accessed: 17.06.2022

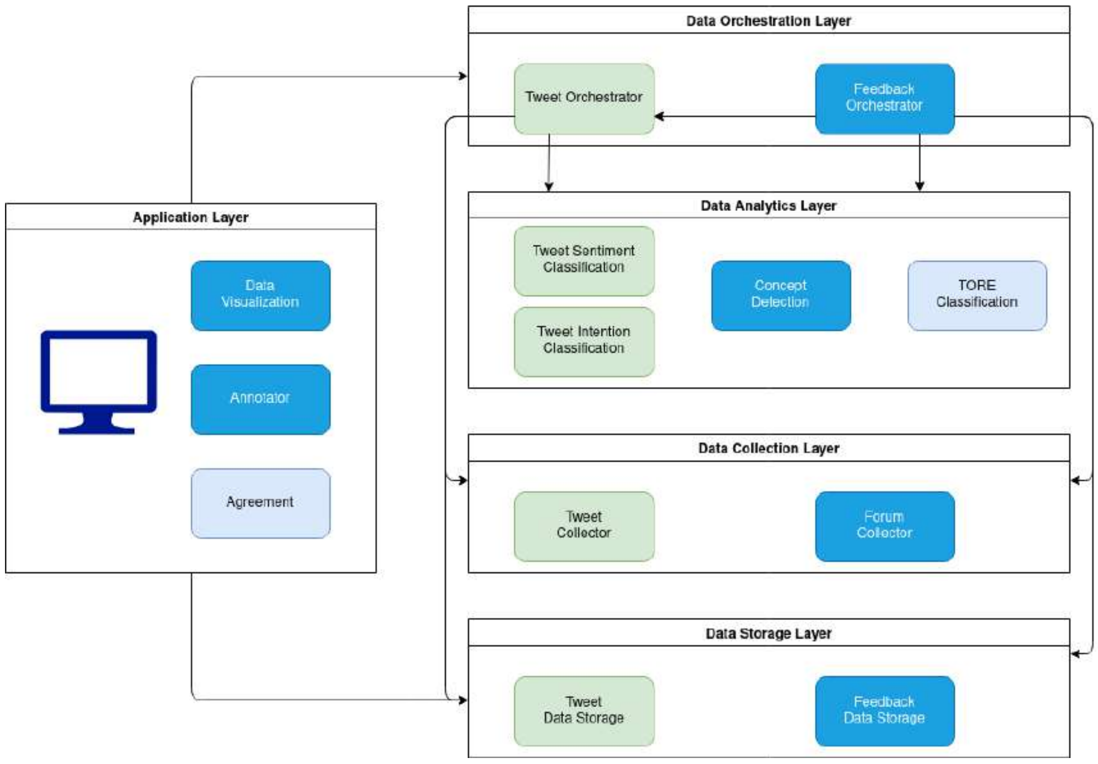


Fig. 2.4: The current architecture of the Feed.UVL project.

2.4.2 Most Relevant Functionalities

Feed.UVL is a tool for the collection and analysis of user feedback. The most relevant features are implemented by the displayed services in figure 2.4, and include:

- The collection, upload and storage of forum data
- The analysis of the datasets with algorithms such as Latent Dirichlet Allocation (LDA) [2], SeaNMF [15] which is a non-negative matrix factorization model, and others
- The visualization of datasets and algorithm results
- The extraction and analysis of problem reports and inquiries from *Twitter*
- Annotation functionalities

Functionalities for annotations include the possibility to create an annotation for a dataset, as well as the possibility of encoding one or more text passages with a word code, a category or a relationship to another text passage. Further, for the encoding process an editor is provided, with which encodings for text passages can be assigned. In a different view insights into the annotations are summarized, such as the number of occurrences of a category, word code or relationship.

2.4.3 Docker & Jenkins

In order to realize the microservice architecture, two tools are used: One for containerization, one for continuous delivery and continuous integration.

Docker

Docker[2] is a self-described "platform for developing, shipping and running applications, and is a tool to support the containerization of applications". *Containers* are small, isolated environments, in which microservices can run. They are independent from other containers, and are shipped as complete units, which are not changed while deployed. A comparison between containers and *Virtual machines* (VMs) can be observed in figure 2.5. Virtual machines run instances of applications, and have their own complete operating systems. Containerized applications use the operating system of the host, making the containers comparatively small.

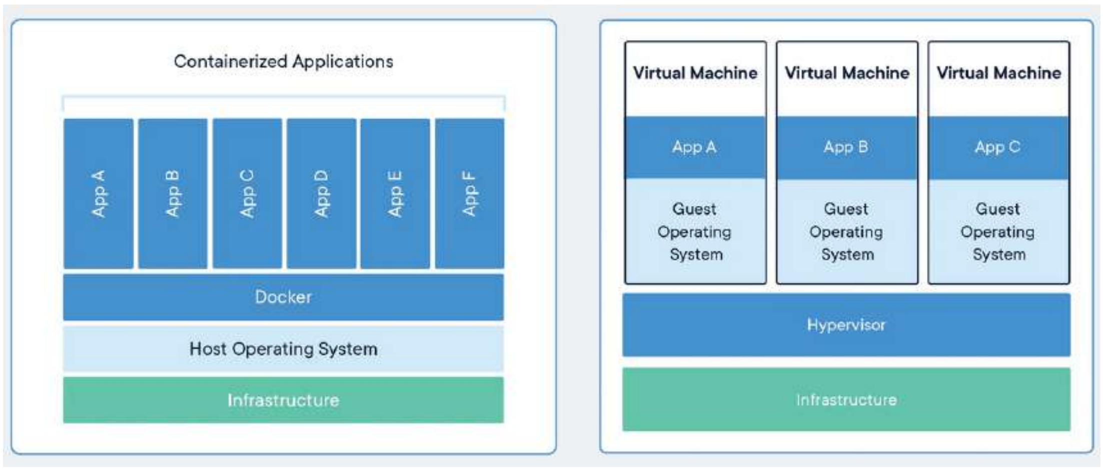


Fig. 2.5: Containers vs. VMs[3].

In Feed.UVL, *Docker* is used to run and manage the microservices.

[2] https://www.docker.com/
[3] https://www.docker.com/resources/what-container/, last accessed: 17.06.2022

Jenkins

Jenkins[4] is a tool used for continuous delivery and integration. In Feed.UVL it is used to build microservices continuously, and it is configured to build and deploy all commits on the master branches of the microservices' *Git*-repositories.

2.5 Bi-LSTM

One of the goals of this thesis is developing a classifier based on a deep learning algorithm. Many of these algorithms consist of neural networks, which are inspired by how the human brain works. The idea is to simulate brain synapses and how they communicate with each other. The basis structure of a neural network can be observed in figure 2.6.

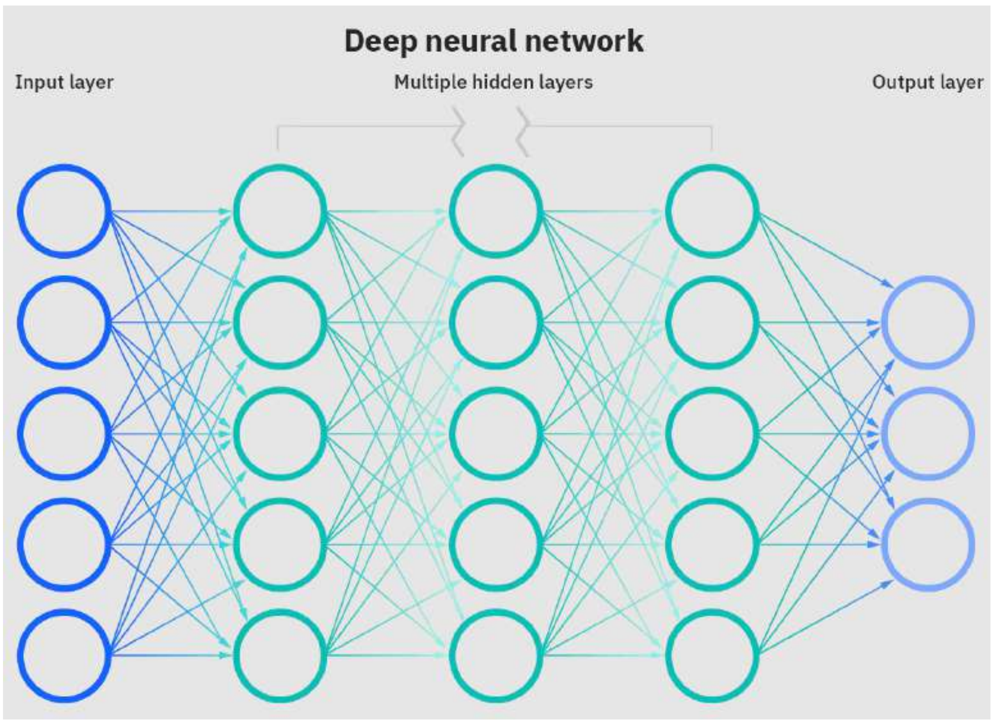


Fig. 2.6: The basic structure of a neural network[5].

The input layer is used to input the data, consisting of feature vectors, into the neural network. There can be at least one hidden layer, which is trained using the training data. Finally, the output layer forms the output for the feature vectors. Neural networks

[4]https://www.jenkins.io/
[5]https://www.ibm.com/cloud/learn/neural-networks, last accessed: 20.06.2022

are trained using weight and bias, which determine which features should be weighted more significantly than others.

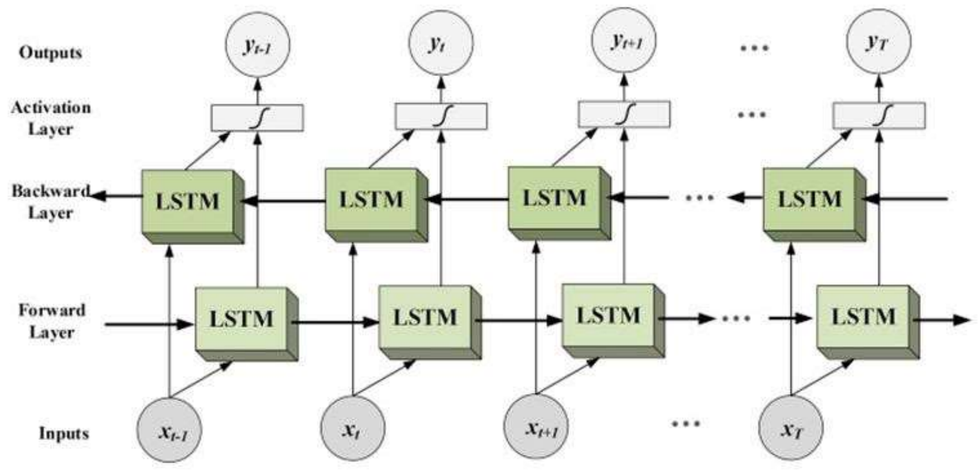


Fig. 2.7: The basic structure of Bi-LSTM[6].

Recurrent Neural Networks (RNNs) are neural networks, which remember the previous words of a text document in order to predict the following words. This is also called *Long Short-Term Memory* (LSTM), as previous words are taken into account in the prediction of new words. *Bidirectional Long Short-Term Memory* (Bi-LSTM) is a combination of two RNNs, which work in opposite directions. A corresponding illustration is provided in figure 2.7. There are two RNNs, one is for forward memory, the other for the backward memory, the combination of which determine the output for the input vectors. Bi-LSTM networks tend to work well for instances, in which understanding the context is useful, because for every word the previous as well as the following words are taken into account.

[6]https://analyticsindiamag.com/complete-guide-to-bidirectional-lstm-with-python-codes/, last accessed: 20.06.2022

3 Literature Review

In this chapter, a literature review on the current state of the art in regards to developing a deep learning algorithm for classifying forum data is summarized. In section 3.1, the exact research question as well as some sub-questions in regards to the content of the papers are stated. In section 3.2 the process of the full literature review is presented, including criteria of relevance, the initial search terms and search results, and the result after snowballing. Afterwards, in section 3.3, the chosen relevant sources are quickly summarized and an explanation of their approach is presented. In section 3.4 a synthesis is conducted, and the approaches in regards to the sub-research questions are evaluated. The results of the evaluation are summarized in section 3.5. Lastly in section 3.6 the most fitting method on which to base the approach of this thesis is chosen.

3.1 Research Question

As a basis for the relevance of papers, the following research question is used:

Research Question. *Which approaches exist that automatically classify natural language user feedback from online forums using deep learning, and which characteristics do they have?*

This question can be split up into multiple criteria, in order to establish search terms for the literature review. It has to be mentioned that only user feedback regarding any kind of software product is taken into consideration, so feedback about different fields of study (e.g. education, health care, etc.) is explicitly excluded.

In order to create benchmarks for later evaluation of the approaches, the following sub-questions have to be answered:

Q1: Which categories for classification are used in the approaches?

Q2: Which deep learning methods are used for which steps of the classification process in the approaches?

Q3: Which pre-processing steps are necessary in order to apply the approaches to the forum data?

Q4: How large are the training datasets used in the approaches and what metadata is necessary?

Sub-question Q1 is relevant, since the number as well as the kind of categories used in the methods can differ from the categories chosen in this thesis. For example, an algorithm that works well on a binary classification problem may not be applicable to a classification with more than two categories. Sub-question Q2 can be important, as different deep learning methods can have different strengths and weaknesses, which may be taken into consideration. It is also important to consider sub-question Q3, as pre-processing of the forum data may be extensive and complex. Lastly, sub-question Q4 is relevant as it is important to determine the amount of forum data which has to be annotated, and whether the necessary metadata even exists in the dataset.

3.2 Research Process

This section includes a detailed summary of the literature review process. The introduction of the criteria of relevance in subsection 3.2.1 is followed by a summary of the initially relevant results after a search-term-based literature search in subsection 3.2.2. The results of snowballing are presented in subsection 3.2.3.

3.2.1 Criteria of Relevance

In order to distinguish the relevancy of papers relatively quickly, some criteria has to be established first. The criteria of relevance is listed as follows:

CoR1: The paper has to be either openly accessible or accessible using the university's credentials.

CoR2: The paper is not older than 7 years.

CoR3: The paper is written in English.

CoR4: The paper is a fully completed research article.

CoR5: Either title or abstract seem relevant in regards to the research question.

CoR6: The paper contains a deep learning method, and considers online forums in which users give feedback for software.

CoR1 to CoR4 contain general criteria, mainly focusing on metadata of an article. This thesis focuses on papers which are not older than seven years, as creating deep learning algorithms to classify data from online forums is still a relatively well-research field, giving a high probability that improvements have been made within the past few years. Also, as many articles were marked as work in progress during the search, a criteria is added necessitating the full completion of articles. CoR5 and CoR6 clarify the relevance of the content of an article. For an initial review, at least either the title or the abstract have to be relevant in regards to the research question, more specifically the article has to contain information about online forums which contain feedback for software.

Using these criteria as a basis, the initial search-term-based approach is conducted.

3.2.2 Initial Search

The first important step in creating feasible search terms for an initial search is identifying synonyms, or terms frequently used as synonyms in the field of study, as well as refinements of those terms. After some searches using IEEE and ACM, synonyms and refinements were developed, as can be observed in table 3.1. Most of the refinements specified are due to terms often used in initially relevant papers, such as *Reddit* in contrast to a lesser known forum, or CNNs in contrast to for example RNNs.

Table 3.1: The synonyms and refinements used for a search-term-based approach.

Base terms	Synonyms and Refinements
classification	text-labeling, requirement mining, feedback mining
natural language	NLP
online forums	Reddit, Stack-Overflow, software product forums, bulletin boards, community question answering
deep learning	AI, neural network, machine learning, CNN

Various combinations of those synonyms were used to create the most successful search terms for the two corpora, IEEE and ACM. Initially, the search via IEEE yielded a larger amount of results, especially when using relatively complex search terms. After a while shorter terms could be used to extract new initially relevant articles.

First attempts of using ACM would prove to be relatively difficult, with searches yielding either too many results, or no results at all. Broadly simplifying the search terms, however, resulted in manageable amounts of articles and could therefore be easily used. In summary, all search terms which yielded at least one new initially relevant result are listed in table 3.2.

Many more combinations of search terms were used on both corpora, but as those yielded no initially relevant results they are excluded here. Formatting differences are due to the use of different query languages by IEEE and ACM. Using the search terms on the two databases, the initially relevant results can be observed in table 3.3.

All papers deemed not relevant were excluded because of CoR5, as only the title and abstract were used as determining factors. Most papers were excluded because they did not use forum data about software, many others were excluded because they did not use a deep learning approach. It has to be mentioned that some of the initial search results may not be completely relevant to the research question, but may contain valuable references which in turn may be relevant. Those are also used in the following snowballing.

3.2.3 Snowballing

From the initially relevant sources extracted in the previous section, snowballing is conducted. Forward snowballing includes inspecting all papers referencing the initially

Table 3.2: The search terms that resulted in at least one new initially relevant result.

Name	**Term**
S1	("Document Title":**classification**) AND ("Document Title":**natural language**) AND ("Document Title":**user feedback**) OR ("Abstract":**user feedback**) AND ("Document Title":**forum**) OR ("Abstract":**forum**) AND ("Document Title":**deep learning**)
S2	("Abstract":**requirement**) AND ("Abstract":**natural language**) AND ("Document Title":**feedback**) OR ("Abstract":**feedback**) AND ("Abstract":**forum***) AND ("Abstract":**unsupervised**)
S3	("Full Text & Metadata":**requirement mining**) AND ("Full Text & Metadata":**natural language**) AND ("Full Text & Metadata":**user feedback**) AND ("Full Text & Metadata":**online product forum***) AND ("Full Text & Metadata":**deep learning**) AND ("Full Text & Metadata":**classif***) AND ("Abstract":**feedback**)
S4	("Abstract":**feedback**) AND ("Abstract":**stack overflow**)
S5	("Abstract":**forum**) AND ("Abstract":**classif***) AND ("Abstract":**deep**)
S6	[Abstract: **deep learning**] AND [Abstract: **user feedback**] AND [Abstract: **classification**] AND [Title: **forum**]
S7	[Abstract: **deep learning**] AND [Abstract: **user feedback**] AND [Abstract: **requirement mining**] AND [Title: **forum**]

relevant source using the criteria of relevance, while backward snowballing takes the papers referenced by the initially relevant source as a basis. Both IEEE and ACM provide functionalities to simplify that process, by providing lists of all citations and references, as well as links to the papers.

Usually not all initially relevant sources are used for snowballing; first they are inspected to establish their relevance, and they are only used in the snowballing if they could be included in the synthesis. But in this review, there were some papers which after further examining the full text did not appear to be completely relevant, but had the potential to reference or be referenced by sources of high relevance. For example, a paper may fit all criteria except for containing a deep learning algorithm, but that was only established after reading the full text, and it used deep learning benchmarks for their comparison or in their related work section. Such a source is not relevant for further analysis, but may yield high quality sources, and is therefore included in this snowballing.

Table 3.3: The initially relevant references after the search-term-based approach.

Library	Term	Date	Results	Initially Relevant	References
IEEE	S1	08.03.2022	33	6	[10], [21], [22], [12], [19], [13]
IEEE	S2	08.03.2022	8	1	[16]
IEEE	S3	08.03.2022	19	2	[20], [4],
IEEE	S4	12.04.2022	13	1	[28]
IEEE	S5	12.04.2022	27	3	[9], [25], [23]
ACM	S6	28.03.2022	14	1	[26]
ACM	S7	04.04.2022	11	1	[13]

The results of snowballing can be observed in table 3.4. For each source, the forward (denoted F) and backward (denoted B) snowballing results are shown. All in all 15 papers were used as a basis for snowballing, but many of those ultimately did not contain any new relevant references. Only those searches which yielded new results are shown.

Most papers in the backward snowballing were filtered out because they were older than seven years. Many others were deemed irrelevant to the research question because they either only took app reviews, *Github* commits or *Twitter* into consideration, or because they did not use deep learning methods. Some papers did not contain user feedback for software.

After evaluating all initially relevant papers, either from the search-term-based search or from snowballing, for their relevance, the results are shown in table 3.5. Reading through the entire contents of the papers, only five seem to completely fulfil the criteria of relevance. The classification methods and category types are relatively diverse, ranging from correct answer prediction to malware detection, and using different kinds of forums as a basis.

3.3 Review Results

An example of binary classification is done by Iftikhar et al. [10]. The dataset considered consists of a combination of questions and answers from *Stack Overflow* with the goal

Table 3.4: The new initially relevant references after snowballing.

Source	Date	F/B	Number of Articles	Initially Relevant	New Papers
[10]	04.04.2022	F	0	0	
		B	51	1	[29]
[22]	11.04.2022	F	9	1	[4]
		B	50	0	
[26]	05.04.2022	F	85	3	[14], [5]. [27]
		B	52	1	[24]
[25]	15.04.2022	F	11	1	[18]
		B	14	1	[8]

of predicting correct answers. The dataset is pre-processed using *Natural Language Processing* (NLP) techniques, keywords are extracted and ranked using TextRank and word embeddings are introduced to get text-based feature vectors. Finally, those vectors are used to train a deep-learning-based ensemble model to predict correct answers, which is based on CNNs as well as LSTMs. This algorithm achieves an accuracy of 84.39%, a precision of 96.16%, a recall of 84.52% and an F1-score of 89.97%.

Xu et al. [26] formulate the problem of linking *Stack Overflow* posts as a multiclass classification problem. The dataset again consists of questions and answers from *Stack Overflow*, the goal is to establish relatedness of different posts via semantically linkable knowledge, which is useful for tasks such as recommendation or duplicate detection. The four classes Xu et al. chose to integrate are *duplicate*, *directly linkable*, *indirectly linkable*, and *isolated*. They use word embeddings generated from data from *Stack Overflow* in combination with Word2Vec, and train the deep learning algorithm, a CNN. Xu et al. achieve an accuracy of 84.1%, a precision of 84.7%, a recall of 84.2% and an F1-score of 84.1% overall.

Another problem is proposed by Li et al. [14], who use the Chinese *Stack Overflow*, CSDN, as the basis for their approach. Their goal is named-entity recognition of software-specific entities, and the classification of those into six different software-related categories. The dataset contains two different languages, Chinese and English, which makes the classification process harder. Li et al. use Word2Vec to generate word embeddings, and combine those with tags such as *Beginning of class*, *End of class*, etc. Together they are used for a Bi-LSTM deep learning algorithm in order to generate the

Table 3.5: The relevant sources after snowballing and filtering.

Authors	**Title**	**Ref.**	**Library**	**Process**
Iftikhar et al.	Deep-Learning-Based Correct Answer Prediction for Developer Forums	[10]	IEEE	Search-Term
Xu et al.	Predicting semantically linkable knowledge in developer online forums via convolutional neural network	[26]	IEEE &ACM	Search-Term
Li et al.	Feature-Specific Named Entity Recognition in Software Development Social Content	[14]	IEEE	Snowballing
Guo et al.	Systematic Comprehension for Developer Reply in Mobile System Forum	[9]	IEEE	Search-Term
Grisham et al.	Identifying mobile malware and key threat actors in online hacker forums for proactive cyber threat intelligence	[8]	IEEE	Snowballing

named-entities and their classifications. The precision achieved by them is 74.805%, the recall is 69.019% and the F1-score is 71.702%. The accuracy is not mentioned.

Guo et al. [9] study replies of developers to user reviews in Chinese mobile forums. The goal is the analysis of the replies, including whether a user review is replied to at all, as well as the reply time. The dataset consists of user reviews which are posted to the forums, and the replies of developers if those are given. It is also completely in Chinese, with the authors mentioning they translated non-Chinese words beforehand. There are two types of classification involved: firstly, binary classification is used to determine whether a review is to be replied to or not. Secondly. the predicted reply time is divided into three classes, *short*, *middle* and *long*. Both problems are solved using Word2Vec for pre-processing, and a CNN-based weak-supervision method for the classification. The authors do not mention the accuracy of their method, but the average precision is 81.7%, the average recall is 82.9% and the F1-score ranges between 77.0% and 83.3%.

A different type of forum is studied by Grisham et al. [8]. They crawl data from hacker forums, in order to find information about malicious hacks and hackers beforehand, thereby increasing cyber security. Their goal is to identify mobile malware as well as key hackers by classifying mobile hardware attachments of forum posts. The dataset consists of the content of the post, title, some additional metadata and attachments, and is crawled from four different kinds of forums in three different languages. The algorithm used for the binary classification is an LSTM RNN classifier. After classification, the author build a network of all hackers who posted an attachment identified as mobile malware, in order to find key actors. This identification, however, is completely separate and not relevant to this paper. The authors achieve a precision of 95%, a recall of 81% and an F1-score of 87%.

3.4 Synthesis

The basis for the synthesis are the research sub-questions as mentioned in section 3.1. Synthesis criteria for comparison are established, in order to evaluate the characteristics of the research questions.

RQ1 can be divided into two synthesis criteria, which are the type of categories and the amount of those. As the forum data used as a basis in this paper is classified into multiple categories, the number of categories used in the relevant sources may be an important indicator as to whether the method could be used. In table 3.6 a short summary of the types of categories can be observed for all sources.

Table 3.6: Comparison of categories for classification (Q1).

Source	**Number of Categories**	**Type of Category**
[10]	2	Correct or incorrect answer for question
[26]	4	Degree and type of relatedness of questions
[14]	6	Software-related categories
[9]	2/3	Whether a review is replied to or not, and how long a reply is predicted to take
[8]	2	Whether an attachment is a mobile malware or not

It can be seen that two sources, [10] and [8], only include binary classification. In both cases, the distinguishing factor is whether an object has a certain property or not. Guo et al. in [9] propose a method which has two different classification stages: First, there is a binary classification process in order to decide whether a review is replied to at all, which is similar to the before mentioned classifications. Second, a reply time is predicted for those review, with three possible categories (*long*, *middle*, *short reply time*).

The sources that only include one classification process and more than two categories for classification are [26] and [14]. In [26] the dataset consists of multiple software-related questions, and all of them are classified in relation to another. The categories are whether they are *duplicates*, *directly linked*, *indirectly linked* or *isolated*, meaning unrelated. [14] supports the largest number of categories, which are all software-related. The dataset consists of forum posts that discuss software, and the categories include *Programming Language*, *Platform*, *API*, *Tool-library Framework*, *Software Standard* and *Undefined Functions*. Also, only the individual words or phrases are classified with this method, as the goal is named-entity recognition.

All in all the number of categories is relatively small in all papers, with many of them using binary classification. However, the classification types are very diverse across the relevant papers, including correct answer, semantic relatedness, software category, reply(-time) and mobile malware classification.

It is possible that multiple deep learning algorithms are used within one paper, or that a deep learning algorithm is used in combination with a non-deep learning method. Therefore it is important to distinguish not only between different kinds of deep learning methods, but also between the steps in the classification approaches. To achieve this analysis, the sub-question RQ2 is divided into three synthesis criteria, which are the step of the algorithm for which any machine learning algorithm is used, which method is used specifically, and lastly whether the method is a deep learning algorithm. The comparison of the sources can be observed in table 3.7.

Table 3.7: Comparison of methods for classification (Q2).

Source	**Step of Algorithm**	**Type of Method**	**Deep Learning**
[10]	Keyword-extraction	TextRank	No
	Word embedding	Word2Vec	No
	m-feature learning	CNN	Yes
	k-feature learning	LSTM	No
	t-feature learning	LSTM	No
[26]	Word embedding	Word2Vec	No
	Semantic-relatedness prediction	CNN	Yes
[14]	Word embedding	Word2Vec	No
	Classification	Bi-LSTM	Yes
[9]	Word embedding	Word2Vec	No
	Reply classification & Time regression	CNN	Yes
[8]	Classification	LSTM RNN	Yes

In [10], many different kinds of algorithms are used for various steps of classification. In order to pre-process the data for the deep learning algorithm, TextRank and Word2Vec are used for keyword-extraction and word embedding. With the help of those, metadata features (*m-features*), keyword features (*k-features*) and word embeddings (*t-features*) are established. They are used as the basis for CNN as well as LSTM learning. The only deep learning algorithm used in this pipeline is the CNN, which uses the m-features as a basis. [26] as well as [9] both use Word2Vec for developing text embeddings for the data, and CNNs for the classification process itself. [9] includes one CNN for both classification tasks, which outputs reply classification as well as time prediction.

In [14] Word2Vec is again used to create text embeddings, but the classification algorithm is a Bi-LSTM deep learning algorithm. [8] is the outlier, as there is no pre-processing algorithm explicitly mentioned. It only includes a deep learning algorithm for the mobile malware classification, an LSTM RNN. All in all most methods used in the sources include creating text embeddings with Word2Vec, and most include some form of neural network. [10] utilizes many different kinds of algorithms, while all others use at most two.

The approaches as presented in the relevant sources have widely varying pre-conditions as well as datasets. Therefore it is necessary to use RQ3 in order to be able to apply the approaches to the forum data. The necessary pre-processing steps can be observed in table 3.8. The most detailed list of pre-processing steps are provided by [10] and [9]. In [10], all pre-processing steps are explicitly mentioned as well as described, and it includes an extensive amount of pre-processing steps. Firstly, many natural language pre-processing steps are applied to the data directly, in order to normalize it and to extract additional data. Then a keyword extraction, ranking algorithm and word embeddings are used to extract features that can be used in the learning pipeline. In [9] there is also a detailed list included, and there are extensive descriptions of all processes except the invalid character extraction. All text has to be translated into one language for this approach to work, which in the paper is Chinese.

In [26] as well as [8], NLP or respectively data pre-processing is mentioned, but not further evaluated on. Additionally, [26] uses word embeddings, and the data has to be rearranged in such a manner that pairs of posts are created which have a certain relationship. In [8], the length of sentences is reduced to 10,000 characters, as the authors claim this speeds up the training process. Lastly, in [14] no natural language pre-processing is mentioned at all, only word embeddings and the removal of any block code in the data.

The last sub-question, RQ4, is necessary to determine the amount of data needed to

Table 3.8: Comparison of necessary pre-processing for classification (Q3).

Source	Necessary pre-processing
[10]	Data Cleaning, Spell Correction, Lowercase Conversion, Tokenization Stop-Word Removal, Stemming, Lemmatization, Keyword Extraction and Ranking, Word Embedding
[26]	Natural language pre-processing (not further specified), Word Embedding, Creation of pairs of posts with user-selected links
[14]	Removing any block code, Word Embedding
[9]	Invalid Character Extraction, Stops Words Removal, Translation of other languages into one language (Chinese), Word Segmentation, Normalization, Word Embedding
[8]	Data pre-processing (not further specified), Cutting sentence length to 10,000 characters

successfully apply the approaches to the forum dataset. As shown in table 3.9, the only synthesis criteria are the size and type of the dataset, as well as the necessary metadata.

Three of the sources have a relatively similar type of dataset, all consisting of posts from either *Stack Overflow* or the Chinese version of *Stack Overflow*. [10] consists of questions and related answers, and needs a comparatively large training dataset, with 91,500 questions and even more related answers. In return, the method only needs one kind of metadata, which are the contained hyperlinks with the information on whether an answer is accepted or not. [26], in contrast, only needs 6,400 pairings of relatedness, meaning a certain amount of posts that create pairs annotated by their semantic relatedness. The exact number of posts is therefor undetermined. The only metadata necessary are the posts links of the *Stack Overflow* post.

In [14], the dataset consists of questions and answers, and entire sub-pages are crawled and parsed to create this dataset. Therefore the amount of 100 web pages equals an amount of 100 questions, each with additional answers. The metadata used in this method is the solution for a question, which is the accepted answer.

A very different type of dataset is used in [8], with posts from four different hacker-forums. The forums are again crawled, and text as well as attachment data is collected. Only the posts with attachments are taken into account, and the size of the training dataset consists of 6,000 of those posts (records). There is extensive metadata used for this approach, because not the text, but the attached metadata is classified. Lastly,

Table 3.9: Comparison of the datasets (Q4).

Source	**Size of Dataset**	**Type of Dataset**	**Necessary Metadata**
[10]	91,500 questions, 236,000 answers	Stack Overflow	Contained hyperlinks (yes/no)
[26]	6,400 pairs, of relatedness	Stack Overflow	Post links
[14]	100 web pages	Chinese Stack Overflow	Solution for a question
[9]	Unknown	Chinese Mobile Forum	Reply information, time tabs
[8]	6,000 records	Hacker-Forums	Attachments, authors, sub-forum name, thread title, post content, attachment name

in [9] the dataset consists of questions and answers from a Chinese mobile forum, with reply information and time tabs as metadata. The size of the training dataset is not mentioned.

A full summary of the synthesis results can be observed in table 3.10.

Table 3.10: Results of the synthesis.

Source	**Number of Categories**	**Type of Category**	**Necessary pre-processing**	**Size of Dataset**	**Type of Dataset**	**Necessary Metadata**	**Step of Algorithm**	**Type of Method**	**Deep Learning**
[10]	2	Correct or incorrect answer for question	Data Cleaning, Spell Correction, Lowercase Conversion, Tokenization Stop-Word Removal, Stemming, Lemmatization, Keyword Extraction and Ranking, Word Embedding	91,500 questions, 236,000 answers	Stack Overflow	Contained hyperlinks	Keyword-extraction Word embedding m-feature learning k-feature learning t-feature learning	TextRank Word2Vec CNN LSTM LSTM	No No Yes No No
[26]	4	Degree and type of relatedness of questions	NLP pre-processing (not further specified), Word Embedding, Creation of pairs of posts with user-selected links	6,400 pairs, of relatedness	Stack Overflow	Post links	Word embedding Semantic-relatedness prediction	Word2Vec CNN	No Yes
[14]	6	Software-related categories	Removing any block code, Word Embedding	100 web pages	Chinese Stack Overflow	Solution for a question	Word embedding Classification	Word2Vec Bi-LSTM	No Yes
[9]	2/3	Whether a review is replied to or not, and how long a reply is predicted to take	Invalid Character Extraction, Stops Words Removal, Translation of other languages into one language (Chinese), Word Segmentation, Normalization, Word Embedding	Unknown	Chinese Mobile Forum	Reply information, time tabs	Word embedding Reply classification & Time regression	Word2Vec CNN	No Yes
[8]	2	Whether an attachment is a mobile malware or not	Data pre-processing (not further specified), Cutting sentence length to 10,000 characters	6,000 records	Hacker-Forums	Attachments, authors, sub-forum name, thread title, post content, attachment name	Classification	LSTM RNN	Yes

3.5 Summary

In this section, the insights from the synthesis are summarized based on the research questions.

Q1: *Which categories for classification are used in the approaches?*

Three of the five approaches use binary classification, although the type of category varies between them. One of those approaches classifies the dataset using a different category at the same time, which makes the model a bit more complex. Two of the sources, [26] and [14], use more than two categories, with one classifying the degree and type of relatedness of forum posts, and the other classifying words into software-related categories. All in all, the types of categories are completely different, showing a high diversity within the research field.

Q2: *Which deep learning methods are used for which steps of the classification process in the approaches?*

All approaches seem to be relatively similar, excluding [8] for which there is not much data available on the pre-processing of the data. The other NLP-based approaches all use word embeddings with the method Word2Vec, and some kind of deep learning algorithm for the classification, with most of them using a neural network. The first source, [10], seems to be a bit more complex, as it utilizes different methods for different features during the classification process. All in all there is not much of a difference between the approaches.

Q3: *Which pre-processing steps are necessary in order to apply the approaches to the forum data?*

All of the approaches require some kind of pre-processing. [10] requires the most, as all pre-processing steps are explicitly listed. Three of the methods require natural language pre-processing, while one needs data processing, and two of the processes are not further specified in the sources. Four of the sources include the usage of word embeddings, while the last, [8], does not.

Q4: *How large are the training datasets used in the approaches and what metadata is necessary?*

The sizes of the datasets vary a lot, from a very large dataset in [10] to smaller sizes for the rest of the approaches. The smallest dataset is used in [14], while no number is available for [9]. The types of the datasets are relatively similar, they are mostly based on some form of *Stack Overflow* or a similar forum. There is only one different type of forum, which is the hacker-forum. The primary concern of this paper is the classification of attachments, which stands in contrast to the other classification methods, which focus on either posts or words within a post. Finally, all approaches use some kind of metadata, which is mostly not too hard to obtain.

3.6 Conclusion

Q: *Which approach should be selected as the basis for this paper?*

In this section, the relevant sources are evaluated based on the suitability of the approaches for the research question and the applicability to the forum dataset, and an approach to base the method on is chosen.

Many of the approaches use binary classification as a basis for their research, which can be problematic for this research question, as there are many different categories involved. Applying a binary classification algorithm to support multiclass classification can be complex. The sources [26], [14] and [9] are the only approaches offering a classification with more than two categories, and may therefore be more applicable. [14] seems to be most significant, as the type of categories is software-related, which is very similar to the software-related categories evaluated in this paper.

The results of RQ2 show that there is not much variability in the kinds of methods used by the sources. Most use Word2Vec to create text embeddings, and some kind of neural network for the classification process. In [10], the model is relatively complex, because different methods are used for varying concerns. The approach is the only one that uses TextRank and LSTM for keyword-extraction and feature learning, respectively. The outliers are formed by [14], which uses a Bi-LSTM model for classification, as well as [8], which has no pre-processing algorithm mentioned and uses an LSTM RNN. This source generally may not be well suited for the classification of the forum data, because the dataset might gain from natural language pre-processing.

Analysing RQ3, the amount of pre-processing varies highly between the approaches, and even after a lot of changes some of the algorithms would not be easily applicable. Nearly all methods require some kind of natural language pre-processing. Additionally, for some approaches there has to be some metadata added or some additional tagging implemented, as the approach has a specific concern. For example for [10] to work, one has to manually create pairings of categories first, in order to find similarities. Similarly with [26], as some semantic links between forum posts have to be created beforehand.

Some sources do not specify any pre-processing, namely [26] and [8], which may be problematic. In [9] the primary language in Chinese, and all non-Chinese words are translated. This could pose problems with the forum dataset, which is in English.

As can be seen from the analysis of RQ4, the sizes of the datasets vary a lot, from a very large dataset in [10] to more reasonable sizes for the rest of the approaches. The smallest dataset is used in [14], while the amount of training data needed is not available for [9]. The types of the datasets are relatively similar, they are mostly based on some form of *Stack Overflow* or a similar forum. There is only one different type of forum, which is the Hacker-forum. As the main concern is primarily about the classification of attachments, this type of dataset seems to be least relevant for the research question of this paper. Finally, all approaches use some kind of metadata, which is mostly not too hard to obtain.

After evaluating all sub-questions as above, the approach of Li et al. [14] seems to be the most fitting as a basis for this research question. It already uses multiple classification categories, needs just some additional metadata and pre-processing, has a relatively small training dataset and has software-related categories.

4 Inter-Rater-Agreement

In this chapter, the inter-rater agreement tool is presented. First, all requirements are provided in section 4.1. Afterwards, the implementation of the requirements, including frontend and backend services, are presented in section 4.2. In section 4.3 a test concept is outlined and the quality of the implementation is assured. Lastly, the implemented tool is evaluated using an informal usability test in section 4.4.

4.1 Requirements

In the following subsections all requirements regarding the inter-rater agreement functionalities are provided. The coarse requirements, from which the functional and non-functional requirements as well as the features were drawn, are presented in subsection 4.1.1. The persona for the coarse requirements is provided in subsection 4.1.2. In subsection 4.1.3 the functional requirements are outlined, followed by a visualization of the relevant domain data in subsection 4.1.4. The non-functional requirements can be found in subsection 4.1.5. Finally, the workspaces for the inter-rater agreement functionalities can be found in subsection 4.1.6, and the corresponding UI-mock-ups are presented in subsection 4.1.7.

4.1.1 Coarse Requirements

The requirements and implementation of the inter-rater agreement is based on the following coarse requirements:

R1 Comparison of annotations: In Feed.UVL, an arbitrary number of annotations for a dataset can be compared. On doing so, in the beginning users can select all annotations of the same dataset which should be compared. Thereafter a new encoding is created, which contains all codes of the previously selected annotations. Users are then able to choose for all text passages, which codes from the selected annotations should be accepted or declined. Additionally, users can have all existing conflicts displayed as a table in another view. Here, a conflict denotes a text passage which is assigned two different codes in two different annotations. Moreover, from this view it is possible to navigate to the corresponding text passage.

R2 Continued use of existing functionalities for encoding: The already existing views and functionalities in Feed.UVL regarding the encoding of datasets are also used in the newly implemented functionalities for the comparison of annotations. This especially refers to the annotation editor, which should also be used for the comparison of annotations, in order to assign codes.

R3 Calculation of the inter-rater agreement: The newly implemented features regarding the comparison of annotations enable users to calculate concrete values (*kappas*), in order to find the consensus between annotators. Available choices should include at least Cohen's kappa, Fleiss' kappa and the kappa of Brennan & Prediger.

4.1.2 Persona

The persona relevant for the inter-rater functionalities is the researcher, which already exists in the Feed.UVL project. The description of this persona already included needs, frustrations and ideal features explaining the use of evaluation and visualization software for natural language statements, which are extended to cover inter-rater agreement requirements.

The biography and knowledge do not need to be changed, as both are already fitting. The needs are extended to cover the requirements as posed in the coarse requirements in section 4.1.1. Some frustrations are added regarding the inter-rater features, especially regarding the UI. Some ideal features are added, which would make the comparison of annotations easier for the researcher. The resulting description of the persona can be observed in table 4.1, with the extended parts of the description in bold, and without the irrelevant parts that already existed.

Table 4.1: The Researcher Persona.

P: Researcher	
Biography	Age 34, doctoral degree in Computer Science, currently researching the user view on software applications. Has previously finished other research projects regarding requirements from software users.
Knowledge	Experience with using evaluation software for natural language texts and using coding tools
Needs	**A possibility for comparing multiple annotations, accepting or declining codes from annotations and seeing some insights into compared annotations. Wants to use the compared annotations for further analysis. All codes for a text passage can be compared, accepted and declined.**
Frustrations	**All progress is lost when an error occurs. It is unclear which encoded text passages have already been resolved and which haven't. It takes a long time to find unresolved codes.**
Ideal Features	**Multiple annotations can be selected to compare annotations. All codes for a text passage can be seen and managed simultaneously. Kappa values are shown. Resolved Comparisons can be exported as new annotations in order to use existing annotation functionalities.**

4.1.3 Functional Requirements

The *Functional requirements* consist of user tasks, subtasks and system functions. User tasks and subtasks are derived from the coarse requirements, and contain refined requirements for the researcher. System functions are functions which realize the requirements of the researcher.

User Tasks and Subtasks

In Feed.UVL, there already exists a user task for the management of TORE annotations. This user task is extended, in order to cover the tasks of the researcher. The new parts of the user task can be seen in table 4.2.

Table 4.2: The User Task.

UT: Manage TORE Annotations
The researcher compares multiple annotations to create an agreement. The researcher resolves disagreements. The researcher analyses the consensus between two annotations. The researcher uses the compared annotations for further analysis.

The user task can be divided into three subtasks. Subtask UTS1 can be derived from coarse requirement R1 and can be observed in table 4.3. Table 4.4 shows the subtask UTS2, which is derived from a combination of coarse requirements R1 and R2. Lastly, in table 4.5 subtask UTS3 can be seen, which is a combination of all three coarse requirements R1, R2 and R3.

Table 4.3: UTS1: Manage Agreement

UTS1: Manage Agreement
1. Create agreement The researcher selects a dataset, selects annotations for that dataset, and picks a name in order to create an agreement. SF12: Create new agreement
2. Inspect agreement The researcher inspects all insights for the agreement, including codes, disagreements and statistics. SF3: Show Code View SF4: Show Agreement Editor View
3. Save agreement The researcher stores his/her progress. SF13: Save Agreement
4. Delete agreement The researcher deletes his/her progress. SF14: Remove Agreement
5. Export Agreement The researcher exports all results for an agreement. SF15: Export Results as CSV

6. Continue working on compared annotations The researcher continues to work on the compared annotation after all disagreements are resolved. SF16: Create new annotation from resolved agreement

Table 4.4: UTS2: Inspect Agreement

UTS2: Inspect Agreement
1. Inspect results of classification algorithms The researcher inspects results of classification algorithms such as part-of-speech and algorithm results. SF18: Highlight Algorithm Results SF19: Highlight POS Tags
2. Calculate Kappa values The researcher calculates initial and current kappa values. SF5: Show Kappa Values SF17: Update Kappa Values
3. Inspect resolved The researcher inspects all resolved codes. SF6: Show resolved codes SF8: Navigate to occurrence

Table 4.5: UTS3: Resolve Disagreements

UTS3: Resolve Disagreements
1. Select a document The researcher selects a document for which to resolve codes. SF20: Show content of document
2. Inspect codes The researcher inspects the codes for a text passage. SF1: Show all codes view SF7: Show unresolved codes

3. Accept code
The researcher accepts a code.
  SF9: Accept code
3a: The researcher creates new code for a passage
  SF10: Create new code
  SF2: Show editor new code view
3p: The user accidentally accepts the wrong code.
  Problem Solution: Revert accepting code
  →Implemented in SF9

---

4. Reject Code
The researcher rejects a code.
  SF11: Reject Code
4p: The user accidentally rejects the wrong code.
  Problem Solution: Revert rejecting code:
  →Implemented in SF9

System Functions

The system functions describe the system's support for the tasks as described in the subtasks. Some of the system functions, such as *SF18: Highlight Algorithm Results*, *SF19: Highlight POS Tags* and *SF20: Show content of document*, are part of the annotation editor, and therefore already exist. As one of the coarse requirements includes the reuse of already existing annotation editor functionalities, only the new system functions will be presented in this thesis.

Navigation functions

Many of the system functions are used for the navigation between different workspaces. As the description of all those navigation functions are very similar, they are only briefly listed in table 4.6.

Table 4.6: All system functions used for the navigation between workspaces.

SF1: Show All Codes View
As a researcher, I want to see all codes for a text passage in order to resolve disagreements.
SF2: Show Editor New Code View
As a researcher, I want to be able to select a category in order to create a new code.
SF3: Show Code View
As a researcher, I want to see all resolved and unresolved codes in order to get an overview of the remaining disagreements.
SF4: Show Agreement Editor View
As a researcher, I want to see the documents in order to find disagreements in the text.
SF5: Show Kappa Values
As a researcher, I want to see initial and current kappa values in order to inspect the consensus between annotators.
SF6: Show Resolved Codes
As a researcher, I want to see all resolved codes in order to navigate to the occurrence in the text.
SF7: Show Unresolved Codes
As a researcher, I want to see all unresolved codes in order to navigate to the occurrence in the text.
SF8: Navigate to Occurrence
As a researcher, I want to navigate to the occurrence of a code in the text in order to analyze the context of the text passage.

SF9: Accept code

As a researcher, I want to accept a code in order to resolve a disagreement.

Table 4.7: SF9: Accept code

SF9: Accept code	
Preconditions	WS10.1: Agreement Editor All Code View or WS9: Agreement Code Results View
Input	Click on "Accept" for code
Postconditions	WS10.1: Agreement Editor All Code View or WS9: Agreement Code Results View
Output	Code marked as accepted and colour is changed to green
Exception	None
Rules	None
Supports	UTS3: Resolve Disagreements

SF10: Create new code

As a researcher, I want to create a new code for a text passage in order to find an alternative for a disagreement.

Table 4.8: SF10: Create new code

SF10: Create new code	
Preconditions	WS10.2: Agreement Editor New Code View
Input	Selected token, word code, category, relationships, other tokens
Postconditions	WS10.1: Agreement Editor All Code View
Output	The new code is created
Exception	Creation of new code is cancelled
Rules	(R1) Either a word code or a category has to be selected
Supports	UTS3: Resolve Disagreements

SF11: Reject Code

As a researcher, I want to reject a code in order to resolve a disagreement.

Table 4.9: SF11: Reject Code

SF11: Reject Code	
Preconditions	WS10.1: Agreement Editor All Code View or WS9: Agreement Code Results View
Input	Click on "Reject" for code
Postconditions	WS10.1: Agreement Editor All Code View or WS9: Agreement Code Results View
Output	Code marked as rejected and colour is changed to red
Exception	None
Rules	None
Supports	UTS3: Resolve Disagreements

SF12: Create new agreement

As a researcher, I want to select a dataset, select annotations for that dataset, and select a name in order to create an agreement.

Table 4.10: SF12: Create new agreement

SF12: Create new agreement	
Preconditions	WS8: Agreement Master View
Input	Dataset, at least two annotations for that dataset, a unique name, boolean for automatic acceptance of codes without disagreements
Postconditions	WS10: Agreement Editor View
Output	An agreement containing all encodings of the annotations
Exception	The name already exists
Rules	(R1) At least two annotations have to be selected.
Supports	UTS1: Manage Agreement

SF13: Save Agreement

As a researcher, I want to save an agreement in order to store my progress.

Table 4.11: SF13: Save Agreement

SF13: Save Agreement	
Preconditions	WS9: Agreement Code Results View or WS10: Agreement Editor View
Input	Input to save annotation, optionally with boolean exit
Postconditions	WS9: Agreement Code Results View or WS10: Agreement Editor View; WS8: Agreement Master View if boolean exit is true
Output	Agreement is saved
Exception	None
Rules	None
Supports	UTS1: Manage Agreement

SF14: Remove Agreement

As a researcher, I want to remove an agreement in order to delete my progress.

Table 4.12: SF14: Remove Agreement

SF14: Remove Agreement	
Preconditions	WS8: Agreement Master View
Input	Agreement
Postconditions	WS8: Agreement Master View
Output	List is without removed agreement
Exception	Removal is cancelled
Rules	None
Supports	UTS1: Manage Agreement

SF15: Export Results as CSV

As a researcher, I want to export my progress as a CSV-file in order to save my results locally.

Table 4.13: SF15: Export Results as CSV

SF15: Export Results as CSV	
Preconditions	WS9: Agreement Code Results View
Input	Click on export button for agreement
Postconditions	WS9: Agreement Code Results View
Output	CSV-file Download
Exception	None
Rules	None
Supports	UTS1: Manage Agreement

SF16: Create new annotation from resolved agreement

As a researcher, I want to create a new annotation from a resolved agreement in order to continue working on the annotation.

Table 4.14: SF16: Create new annotation from resolved agreement

SF16: Create new annotation from resolved agreement	
Preconditions	WS9: Agreement Code Results View
Input	A unique name for the annotation, an agreement
Postconditions	WS9: Agreement Code Results View
Output	New annotation is created
Exception	The name already exists
Rules	None
Supports	UTS1: Manage Agreement

SF17: Update Kappa Values

As a researcher, I want to update the kappa values of an agreement in order to see the current consensus between the annotations.

Table 4.15: SF17: Update Kappa Values

SF17: Update Kappa Values	
Preconditions	WS9: Agreement Code Results View
Input	Click on update button for agreement
Postconditions	WS9: Agreement Code Results View
Output	The current kappa values
Exception	None
Rules	None
Supports	UTS2: Inspect Agreements

4.1.4 Domain Data

The domain for the agreement functionalities in Feed.UVL is depicted in the domain data model in figure 4.1. The agreement consists of two or more annotations and resolved and unresolved disagreements, from which the kappa values can be calculated. An annotation has codes, which has tokens that are based on text passages. The text passages are part of a document, which can be analyzed by algorithms. The documents are part of a dataset, on which annotations for the agreement are based.

4.1.5 Non-Functional Requirements

Functional requirements are requirements that define what the system should support. They usually describe features that should be implemented in the system, and consist of concrete definitions of functions. On the other hand, *Non-functional requirements* (NFRs) describe how well the system supports what the user wants, or more generally how well the system works. They are often related to the quality of the system. NFRs often consist of requirements, for which functions and data are not well defined, nevertheless it should be possible to measure them in some form, in order to find out whether they are fulfilled.

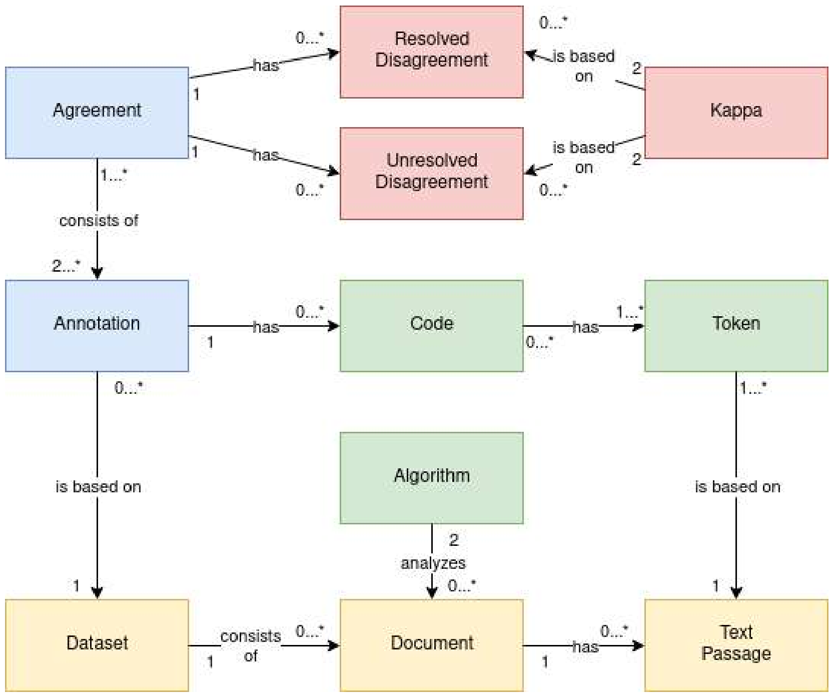


Fig. 4.1: The Domain Data Model for Agreements in Feed.UVL

The standard for NFRs in this thesis is the ISO/IEC 25010 system and software quality standard from [11]. It consists of categories with which to measure product quality, which are themselves further divided into multiple sub-categories. In this thesis, four of those categories are used: *Performance*, *Usability*, *Maintainability* and *Functionality*. All following definitions are taken from this ISO standard[1], all metrics are requirements for the Feed.UVL system.

NFR1: Performance

This characteristic represents the performance relative to the amount of resources used under stated conditions.

Time behaviour

Degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements.

[1] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010, last accessed: 30.06.2022

Metrics:

- When selecting a token in the *WS10: Agreement Editor View*, the *WS10.1: Agreement All Codes View* is shown in less than 100ms.
- When creating a new code for a token, saving the new code takes less than 100ms.
- Creating an agreement without automatically resolving disagreements takes less than 3 seconds per 1,000 tokens.

All performance metrics were tested and confirmed after the complete implementation and quality assurance. The last metric was tested with a dataset containing 1,430 tokens, which took less than 3 seconds, and another dataset consisting of 13,775 tokens, which took less than 40 seconds.

NFR2: Usability

Degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

User error protection

Degree to which a system protects users against making errors.

Metrics:

- All system functions have stoppers, in order to protect from faulty user inputs.
- Deletion of an agreement takes at least two clicks.

Both metrics are assured with system tests, which are included in subsection 4.3.4.

Operability

Degree to which a product or system has attributes that make it easy to operate and control.

Metrics:

- When as user is in the agreement editor view, a code is able to be accepted or rejected with at most two clicks.
- The user can navigate from the list of unresolved codes to the occurrence in the dataset with one click.

The operability metrics are tested during the evaluation of the inter-rater agreement tool, in section 4.4.

NFR3: Maintainability

This characteristic represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements.

Modularity

Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

Metrics:

- The system is split into independent microservices, divided by well defined concerns.

The modularity metric is assured through the implementation using microservices, as documented in subsection 4.2.1.

NFR4: Functionality

This characteristic represents the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions.

Functional completeness

Degree to which the set of functions covers all the specified tasks and user objectives.

Metrics:

- The system functions are covering all the user tasks and subtasks. User tasks and subtasks are based on the coarse requirements. The System functions are derived from the subtasks, thus they can be traced back to them. Each subtask has at least one system function that supports it.

The metric of functional completeness is already assured, as all user tasks and subtasks are based on the coarse requirements as presented in subsection 4.1.1. All subtasks in subsection 4.1.3 contain multiple system functions, which in turn have links to their respective subtasks.

Functional correctness

Degree to which a product or system provides the correct results with the needed degree of precision.

Metrics:

- Every system functions' correctness is verified with system tests. If all system tests pass for a system function, it indicates its correctness. Additionally, the system functions are verified with an informal usability test.

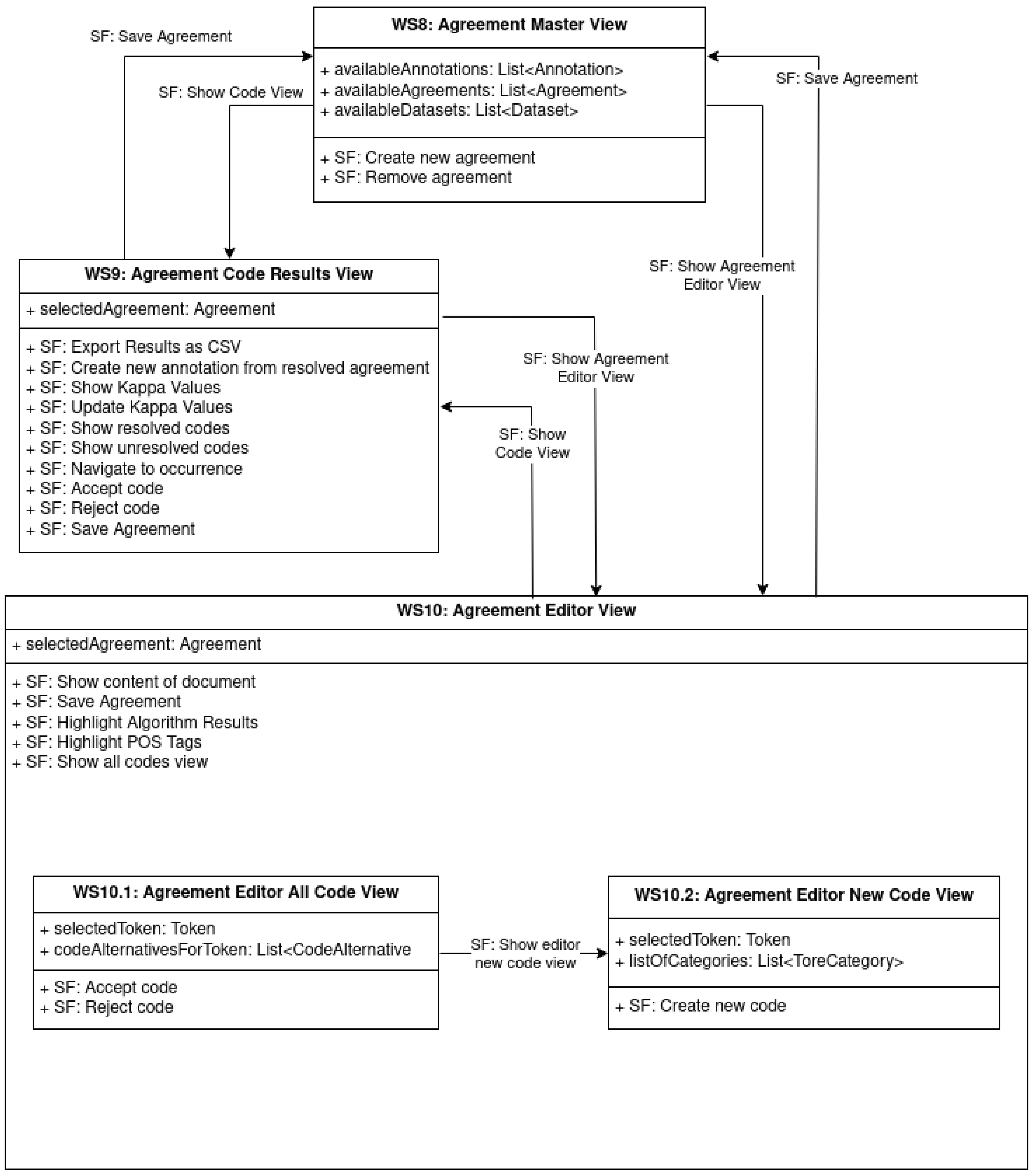
The functional correctness is assured by the system tests as documented in subsection 4.3.4 and verified by the evaluation in section 4.4 using a usability test.

4.1.6 Workspaces

Workspaces (WS) are areas, in which system functions are grouped and tied into a logical context. The UI-structure diagram, illustrated in figure 4.2, connects the workspaces with each other and ties the functionalities together. Only system functions and workspaces regarding agreements are discussed here.

The *WS8: Agreement Master View* is the main workspace. From here agreements can be created, deleted and managed. Navigation to either the *WS10: Agreement Editor View* or the *WS9: Agreement Code Results View* is possible, and navigation between those two workspaces is also supported. In the *WS9: Agreement Code Results View*, all statistics about the agreement, as well as lists of unresolved and resolved codes, are given. In the *WS10: Agreement Editor View* all tokens are shown, navigation between documents is possible, and tokens which are encoded can be selected.

From the previous workspace navigation to the *WS10.1: Agreement Editor All Code View* is possible. Here, all codes for a selected token are listed, and codes can be accepted or declined. One can further navigate to the *WS10.2: Agreement Editor New Code View*, where new codes for the selected token can be created.



WS8: Agreement Master View
+ availableAnnotations: List<Annotation>
+ availableAgreements: List<Agreement>
+ availableDatasets: List<Dataset>
+ SF: Create new agreement
+ SF: Remove agreement
SF: Save Agreement
SF: Show Code View
SF: Save Agreement
SF: Show Agreement Editor View
WS9: Agreement Code Results View
+ selectedAgreement: Agreement
+ SF: Export Results as CSV
+ SF: Create new annotation from resolved agreement
+ SF: Show Kappa Values
+ SF: Update Kappa Values
+ SF: Show resolved codes
+ SF: Show unresolved codes
+ SF: Navigate to occurrence
+ SF: Accept code
+ SF: Reject code
+ SF: Save Agreement
SF: Show Agreement Editor View
SF: Show Code View
WS10: Agreement Editor View
+ selectedAgreement: Agreement
+ SF: Show content of document
+ SF: Save Agreement
+ SF: Highlight Algorithm Results
+ SF: Highlight POS Tags
+ SF: Show all codes view
WS10.1: Agreement Editor All Code View
+ selectedToken: Token
+ codeAlternativesForToken: List<CodeAlternative
+ SF: Accept code
+ SF: Reject code
SF: Show editor new code view
WS10.2: Agreement Editor New Code View
+ selectedToken: Token
+ listOfCategories: List<ToreCategory>
+ SF: Create new code

Fig. 4.2: The UI-structure diagram for the agreement workspaces in Feed.UVL

4.1.7 Mock-Ups

Mock-ups are preliminary illustrations of the system, and are based on the workspaces of section 4.1.6. They are used to get an idea of what the realization of the workspaces and system functions may look like, and can be changed in the final implementation.

Figure 4.3 shows the mock-up for the creation of a new agreement, which corresponds to part of WS8.

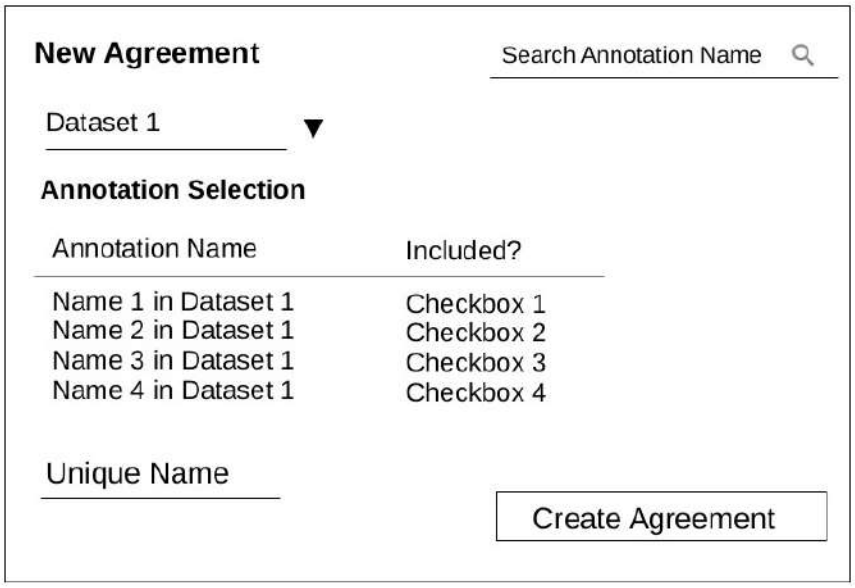
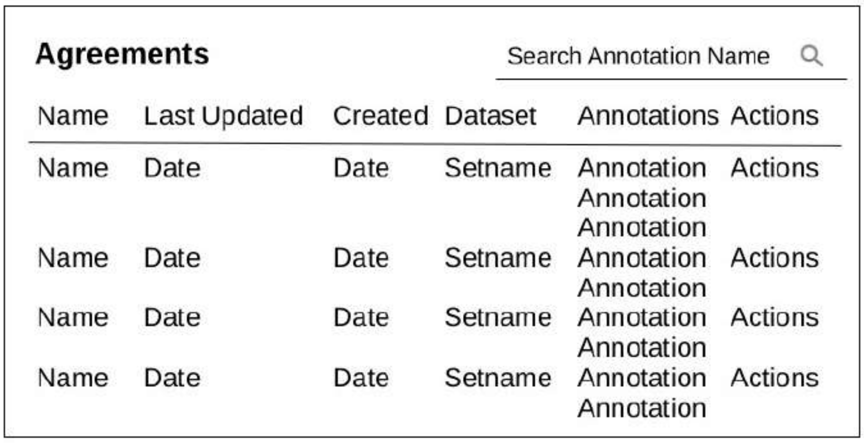


Fig. 4.3: The mock-up for the creation of a new agreement in WS8

In figure 4.3 an illustration of the list of agreements can be seen, from which the deletion or the navigation to other workspaces can be triggered. This is also part of WS8.

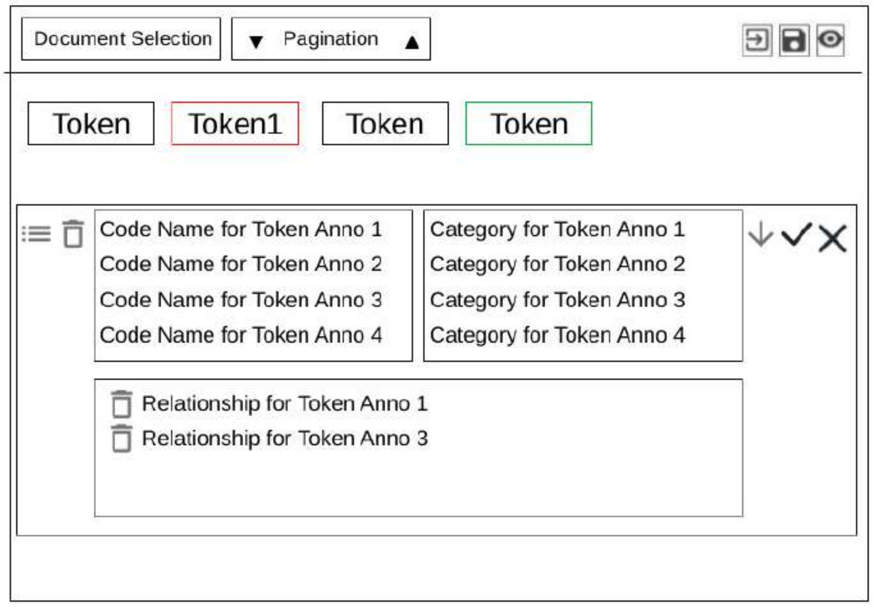

Agreements

Search Annotation Name

Name	Last Updated	Created	Dataset	Annotations	Actions
Name	Date	Date	Setname	Annotation Annotation Annotation	Actions
Name	Date	Date	Setname	Annotation Annotation	Actions
Name	Date	Date	Setname	Annotation Annotation	Actions
Name	Date	Date	Setname	Annotation Annotation	Actions

Fig. 4.4: The mock-up for the management of existing agreements in WS8

An illustration of the agreement editor with a corresponding list of codes can be observed in figure 4.5. The view of the codes for a token in the final implementation looks very different from this mock-up, which shows that this is only one idea for the realization of the workspaces WS10, WS10.1, WS10.2.



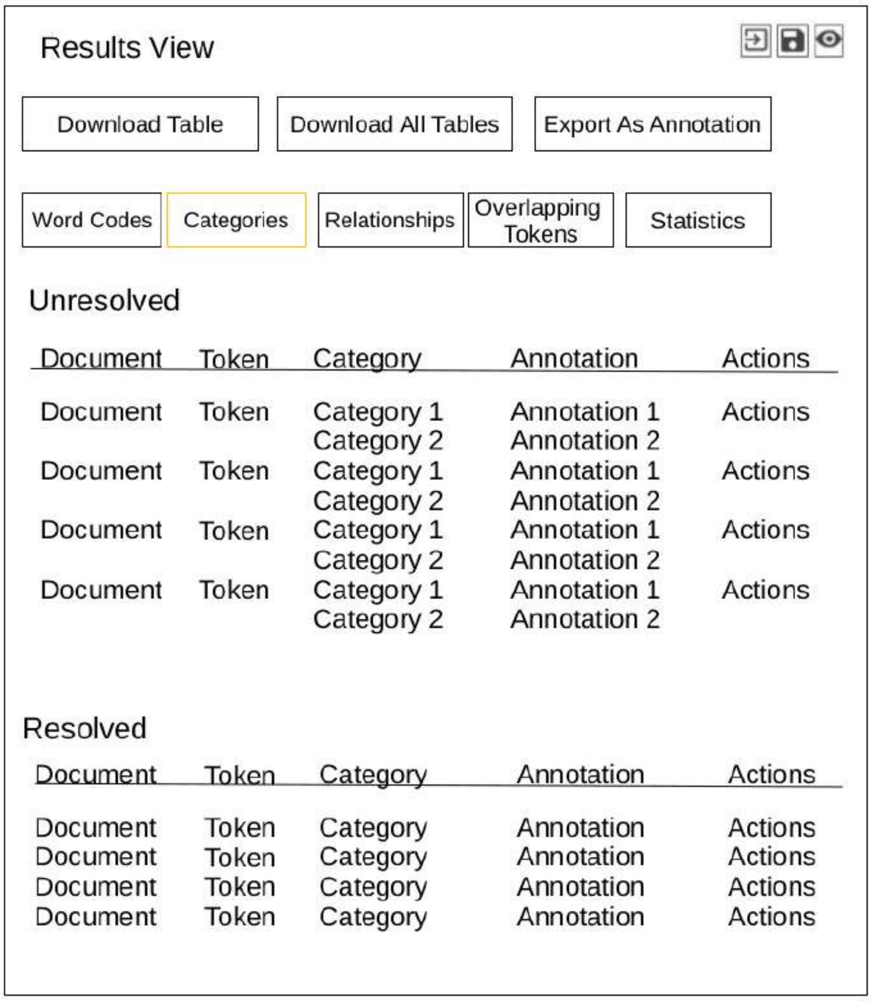
Code Name for Token Anno 1
Code Name for Token Anno 2
Code Name for Token Anno 3
Code Name for Token Anno 4

Category for Token Anno 1
Category for Token Anno 2
Category for Token Anno 3
Category for Token Anno 4

Relationship for Token Anno 1
Relationship for Token Anno 3

Fig. 4.5: The mock-up of the agreement editor, including the list of tokens in WS10, WS10.1, WS10.2

Lastly, in figure 4.6 an illustration of WS9 is shown. There are different kinds of tabs which can be chosen from, as well as lists with resolved and unresolved codes.



Unresolved

Document	Token	Category	Annotation	Actions
Document	Token	Category 1	Annotation 1	Actions
		Category 2	Annotation 2	
Document	Token	Category 1	Annotation 1	Actions
		Category 2	Annotation 2	
Document	Token	Category 1	Annotation 1	Actions
		Category 2	Annotation 2	
Document	Token	Category 1	Annotation 1	Actions
		Category 2	Annotation 2	

Resolved

Document	Token	Category	Annotation	Actions
Document	Token	Category	Annotation	Actions
Document	Token	Category	Annotation	Actions
Document	Token	Category	Annotation	Actions
Document	Token	Category	Annotation	Actions

Fig. 4.6: The mock-up of the results view of an agreement in WS9

4.2 Implementation

In this section the implementation of the agreement functionalities into the Feed.UVL system is presented. In subsection 4.2.1 the microservice architecture of the implementation is discussed. The data classes used in the microservices are outlined in subsection 4.2.2. In subsection 4.2.3 the new and extended backend services are shown, while the frontend services are presented in subsection 4.2.4.

4.2.1 Architecture

The existing microservice architecture was already presented in section 2.4.1. Following the *NFR3: Maintainability*, the microservice architecture is extended with the addition of a new microservice called *uvl-agreement*. The new microservice is part of the application layer, and handles everything related to building an agreement, including the creation of agreements, the conversion to annotation and the calculation of kappas.

Additionally, the existing microservices *ri-visualization*, *uvl-storage-concepts* and *uvl-orchestration-concepts* are extended. The service *uvl-orchestration-concepts* is used to orchestrate the creation of agreements, and communicates with the *uvl-agreement* service as well as the *uvl-storage-concepts* service. The database management system *MongoDB*[2] is running as a microservice itself, while the *uvl-storage-concepts* handles the communication with the database, as well as saving and retrieving data. The *ri-visualization* service handles the frontend, and is extended by the new agreement Views.

All of those microservices are written in the programming language *Go*[3] except for the frontend microservice *ri-visualization*, which is in *Javascript*[4] and uses the *VueJS*[5] framework. All of them are deployed using *Docker* as a containerization tool, and *Jenkins*, which automatically builds and deploys new commits from the respective *Git* repositories.

4.2.2 Data Classes

The data classes implemented by the mircoservices are derived from the domain data model in section 4.1.4, and can be observed in figure 4.7. The data model is used by all services, but the implementation depends on the programming language, as the backend services use *Go*, while the frontend service uses *Javascript*.

[2]https://www.mongodb.com/
[3]https://go.dev/
[4]https://devdocs.io/javascript/
[5]https://vuejs.org/

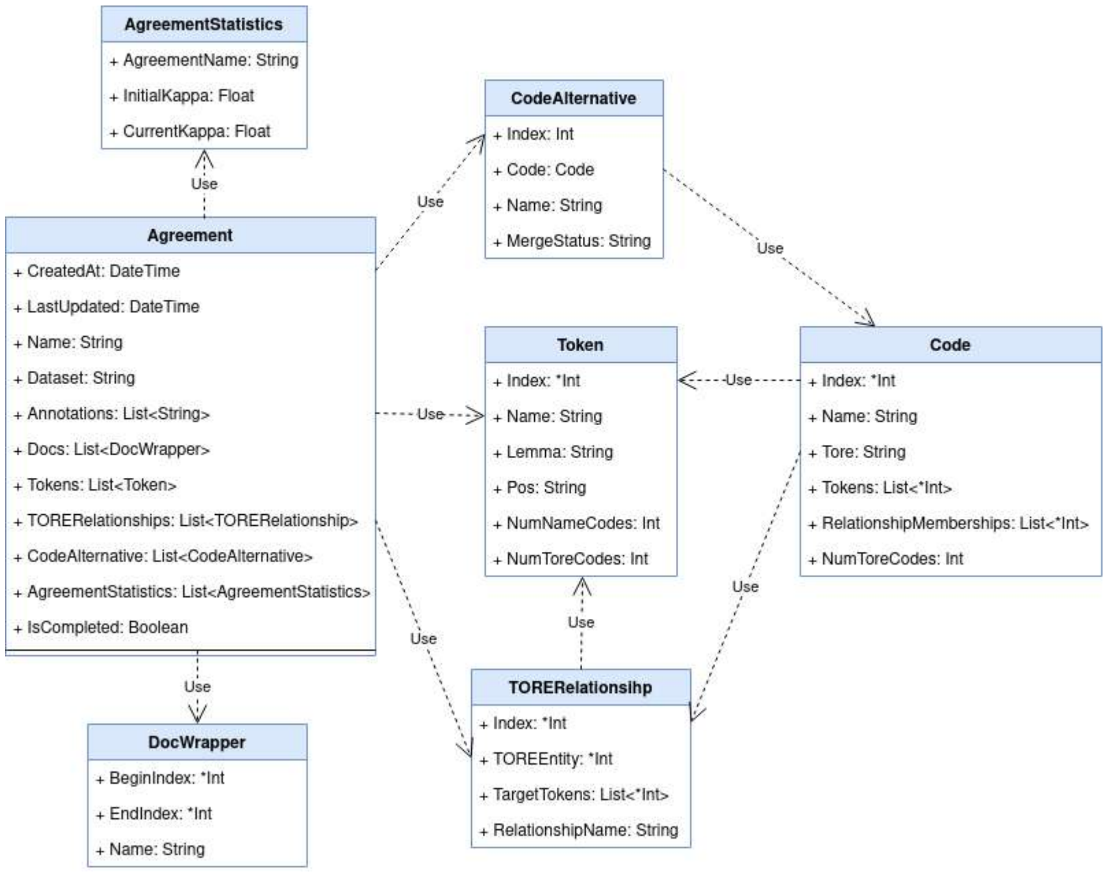


Fig. 4.7: The data classes for all agreement functionalities.

To create an agreement, all selected annotations are compared and the data is merged. The *DocWrapper*-class is the same for all annotations of a dataset, and can be carried over from any of the annotations. The relevant fields of the *Token*-class for agreements are *index*, *name*, *lemma*, and *pos*, which all stay the same for all annotations of a dataset, while *numNameCodes* and *numToreCodes* are not used for the agreement implementation. Therefore the list of tokens can again be taken from any of the annotations, and does not have to be merged or changed.

The *Agreement*-class is the most important class, as it links to all the relevant information of an agreement. *AgreementStatistics* contains the name of a kappa and initial as well as current kappa values. As of now, two types of kappas exist, but in the future more can be added easily. *TORERelationship* contains all relationships of all annotations, and is merged during agreement creation, by assigning unique indices. Similarly, the *Code*-class contains all codes of all annotations and is created by merging during creation and assigning unique indices. *CodeAlternatives* serves as a wrapper around the codes, and carries information about the name of the annotation and the merge status of the code. A merge status can be either *Accepted*, *Pending* or *Declined.* If any pending code is assigned to a token, the token has a disagreement.

4.2.3 Backend Services

Two of the already existing services supporting agreement functionalities in the Feed.UVL project, which are the orchestration service and the storage service, are presented next. Afterwards, the only newly created microservice, the agreement service, is shown. Since all services are written in *Go* in which classes don't exist, all classes in the subsequent class diagrams are only abstract.

Orchestrator and Storage Service

The orchestration service serves as an API for the orchestration of new agreements, the update of the current kappa values, and the export of an agreement as an annotation. All incoming API calls are handled and sent to the responsible services via a RestHandler. The storage service is responsible for storing, extracting and deleting data from the database, *MongoDB*. Other services use API calls to communicate with the database through this service. As both, the orchestrator and the storage service already existed in the Feed.UVL project, only new functions are shown in figure 4.8. The *RestHandler* is implemented in the orchestrator, while the *MongoHandler* is implemented in the storage service.

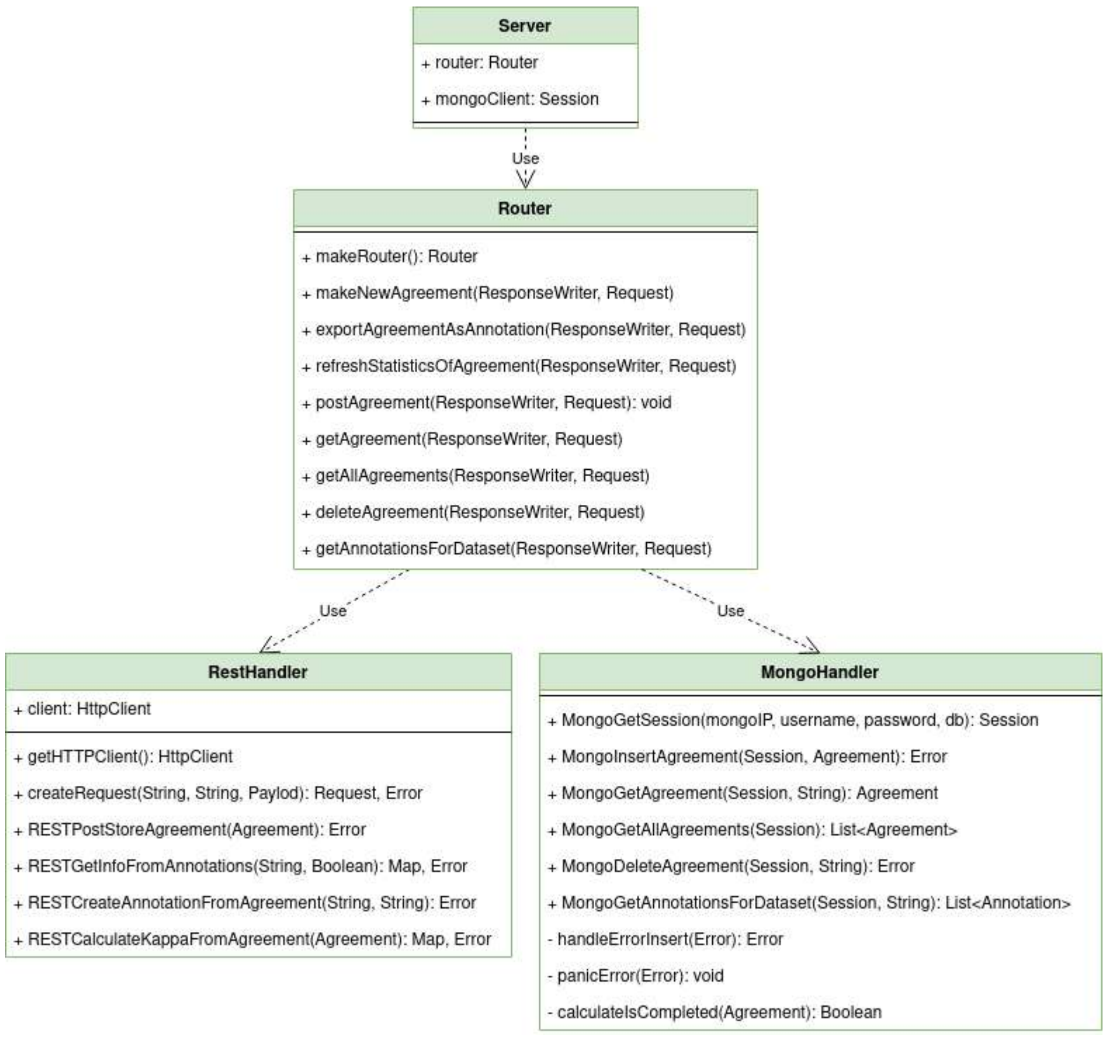

Server
+ router: Router
+ mongoClient: Session
Use
Router
+ makeRouter(): Router
+ makeNewAgreement(ResponseWriter, Request)
+ exportAgreementAsAnnotation(ResponseWriter, Request)
+ refreshStatisticsOfAgreement(ResponseWriter, Request)
+ postAgreement(ResponseWriter, Request): void
+ getAgreement(ResponseWriter, Request)
+ getAllAgreements(ResponseWriter, Request)
+ deleteAgreement(ResponseWriter, Request)
+ getAnnotationsForDataset(ResponseWriter, Request)
Use
Use
RestHandler
+ client: HttpClient
+ getHTTPClient(): HttpClient
+ createRequest(String, String, Paylod): Request, Error
+ RESTPostStoreAgreement(Agreement): Error
+ RESTGetInfoFromAnnotations(String, Boolean): Map, Error
+ RESTCreateAnnotationFromAgreement(String, String): Error
+ RESTCalculateKappaFromAgreement(Agreement): Map, Error
MongoHandler
+ MongoGetSession(mongoIP, username, password, db): Session
+ MongoInsertAgreement(Session, Agreement): Error
+ MongoGetAgreement(Session, String): Agreement
+ MongoGetAllAgreements(Session): List<Agreement>
+ MongoDeleteAgreement(Session, String): Error
+ MongoGetAnnotationsForDataset(Session, String): List<Annotation>
- handleErrorInsert(Error): Error
- panicError(Error): void
- calculateIsCompleted(Agreement): Boolean

Fig. 4.8: The class diagram for the added functionalities in the uvl-orchestration-concepts and uvl-storage-concepts microservice.

Agreement Service

The agreement service is responsible for all complex calculations regarding agreement data. The main functions are the creation of a new agreement, the calculation of the kappa values, and the export of agreements as annotations. It handles API calls which are used by the orchestration service, and has a *RestHandler* to communicate with the storage service in order to use the database. Tasks of the *RestHandler* includes getting information about agreements, annotations, TORE categories and TORE relationships, as well as storing newly created annotations. The class diagram of the annotation service can be seen in figure 4.9.

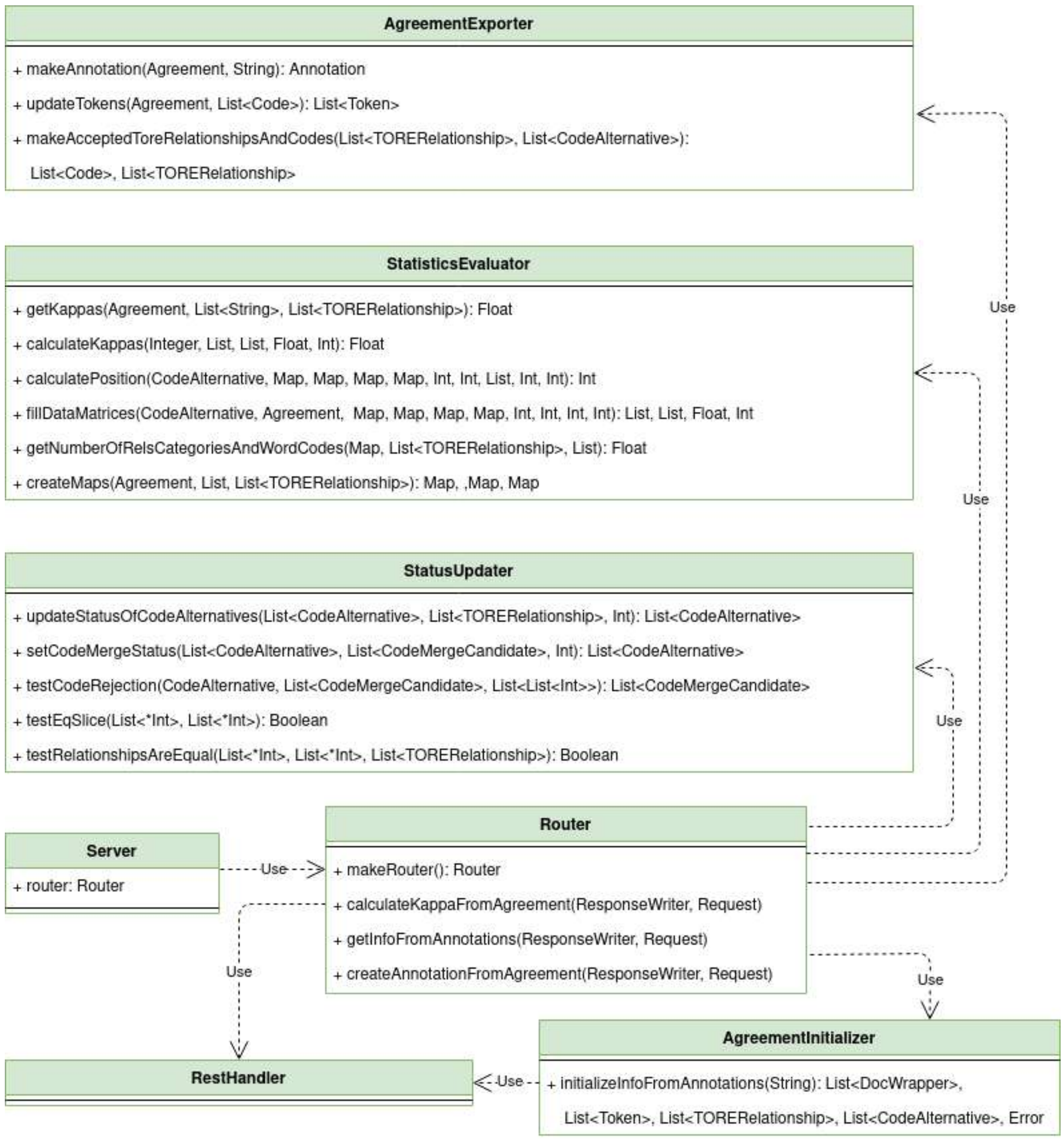

AgreementExporter
+ makeAnnotation(Agreement, String): Annotation
+ updateTokens(Agreement, List<Code>): List<Token>
+ makeAcceptedToreRelationshipsAndCodes(List<TORERelationship>, List<CodeAlternative>): List<Code>, List<TORERelationship>
StatisticsEvaluator
+ getKappas(Agreement, List<String>, List<TORERelationship>): Float
+ calculateKappas(Integer, List, List, Float, Int): Float
+ calculatePosition(CodeAlternative, Map, Map, Map, Map, Int, Int, List, Int, Int): Int
+ fillDataMatrices(CodeAlternative, Agreement, Map, Map, Map, Map, Int, Int, Int, Int): List, List, Float, Int
+ getNumberOfRelsCategoriesAndWordCodes(Map, List<TORERelationship>, List): Float
+ createMaps(Agreement, List, List<TORERelationship>): Map, ,Map, Map
StatusUpdater
+ updateStatusOfCodeAlternatives(List<CodeAlternative>, List<TORERelationship>, Int): List<CodeAlternative>
+ setCodeMergeStatus(List<CodeAlternative>, List<CodeMergeCandidate>, Int): List<CodeAlternative>
+ testCodeRejection(CodeAlternative, List<CodeMergeCandidate>, List<List<Int>>): List<CodeMergeCandidate>
+ testEqSlice(List<*Int>, List<*Int>): Boolean
+ testRelationshipsAreEqual(List<*Int>, List<*Int>, List<TORERelationship>): Boolean
Server
+ router: Router
Router
+ makeRouter(): Router
+ calculateKappaFromAgreement(ResponseWriter, Request)
+ getInfoFromAnnotations(ResponseWriter, Request)
+ createAnnotationFromAgreement(ResponseWriter, Request)
AgreementInitializer
+ initializeInfoFromAnnotations(String): List<DocWrapper>, List<Token>, List<TORERelationship>, List<CodeAlternative>, Error
RestHandler
Use
Use
Use
Use
Use
Use
Use

Fig. 4.9: The class diagram for the uvl-agreement microservice.

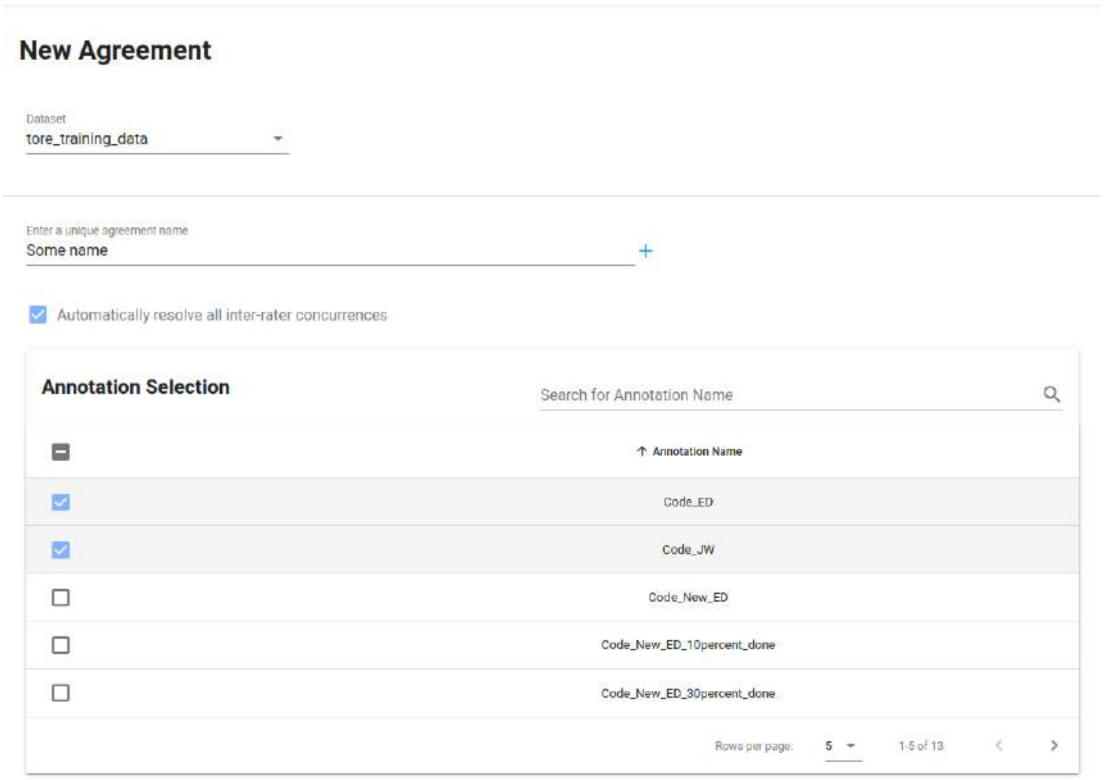
Instead of creating a new agreement service, the *Annotator*-microservice could have been reused as an alternative, as the data structures behind the annotation and agreement are relatively similar. The decision of creating a new microservice was made, since the comparison of annotations is a different concern than simply creating a tokenization of a dataset. Additionally, the agreement service contains multiple features regarding agreements, which justifies the creation of a new microservice. Last but not least, by creating a new service another programming language can be used easily. In order to stay consistent, *Go* is used for the agreement service instead of *Python*, as the orchestration and storage services are both written in *Go*.

4.2.4 Frontend Services

The realization of the workspaces are called views. The microservice *ri-visualization* handles all functionalities related to the frontend, therefore all views have to be implemented in this service.

Agreement Master View

The *Agreement Master View* realizes WS8, and consists of two major features. A dataset can be selected, and a name can be entered. Afterwards a list of annotations for the selected dataset is displayed, which can be seen in figure 4.10. When at least two of those annotations are selected, the "+"-Icon is enabled, and the system function for the creation of a new agreement can be used.



New Agreement

Dataset: tore_training_data

Enter a unique agreement name: Some name

✓ Automatically resolve all inter-rater concurrences

Annotation Selection

Search for Annotation Name

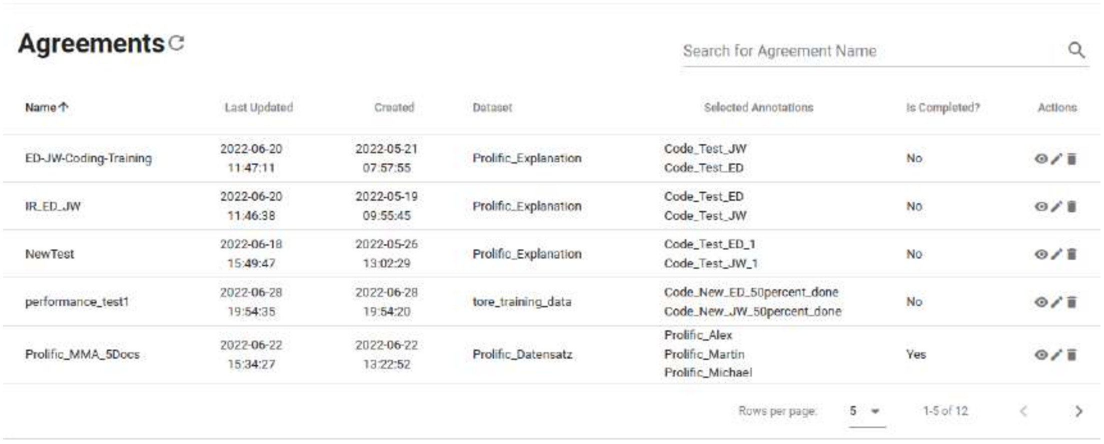
	Annotation Name
✓	Code_ED
✓	Code_JW
☐	Code_New_ED
☐	Code_New_ED_10percent_done
☐	Code_New_ED_30percent_done

Rows per page: 5 1-5 of 13

Fig. 4.10: The view for the creation of a new agreement.

While the design of the realization is different from the mock-up as presented in figures 4.3 and 4.4 in section 4.1.7, the basic functionalities remain in a relatively similar position. One addition is the checkbox to automatically resolve all inter-rater concurrences, which is checked by default. This design decision was made, because it reduces the amount of work when comparing annotations, especially if the kappa values of the inter-rater agreements are high. The alternative would have been an opt-in feature, which, when forgotten about, leads to higher amounts of work or longer waiting times, as creating an agreement can take some time. This could ultimately lead to frustrations for users.

The second feature is the display of all agreements, as shown in figures 4.11. All existing agreements are shown, with an option to delete the selected agreement or to show either the code view or the editor view. The implementation looks very similar to the mock-up.



Agreements

Name ↑	Last Updated	Created	Dataset	Selected Annotations	Is Completed?	Actions
ED-JW-Coding-Training	2022-06-20 11:47:11	2022-05-21 07:57:55	Prolific_Explanation	Code_Test_JW Code_Test_ED	No	
IR_ED_JW	2022-06-20 11:46:38	2022-05-19 09:55:45	Prolific_Explanation	Code_Test_ED Code_Test_JW	No	
NewTest	2022-06-18 15:49:47	2022-05-26 13:02:29	Prolific_Explanation	Code_Test_ED_1 Code_Test_JW_1	No	
performance_test1	2022-06-28 19:54:35	2022-06-28 19:54:20	tore_training_data	Code_New_ED_50percent_done Code_New_JW_50percent_done	No	
Prolific_MMA_5Docs	2022-06-22 15:34:27	2022-06-22 13:22:52	Prolific_Datensatz	Prolific_Alex Prolific_Martin Prolific_Michael	Yes	

Fig. 4.11: The view of all existing agreements.

Agreement Code Results View

The *Agreement Code Results View* realizes WS9, and consists of a header and three tabs, in which information about the agreement is displayed. The header of this View is shown in figure 4.12, which contains the system functions to export the agreement as a CSV. When an agreement is completely resolved, there is also the possibility to convert the agreement to an annotation.

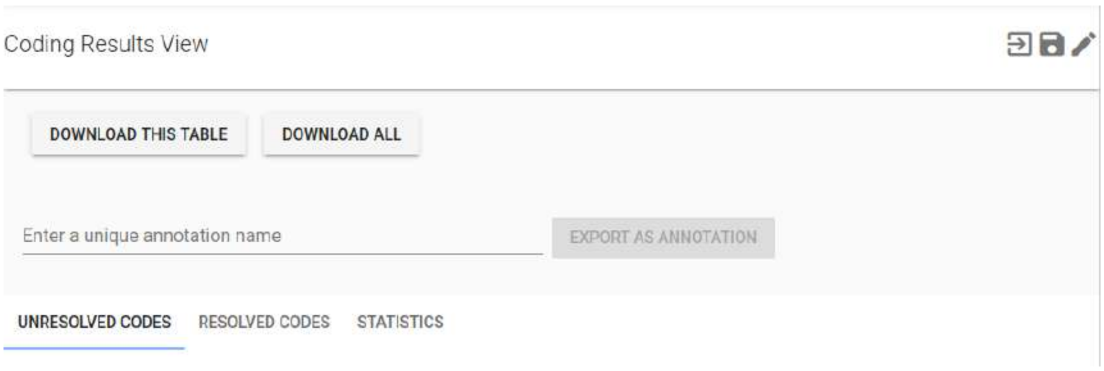
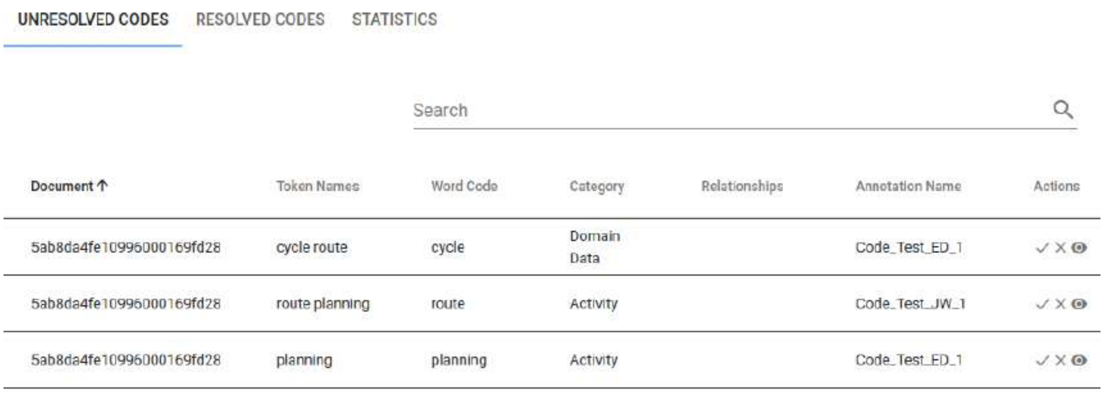


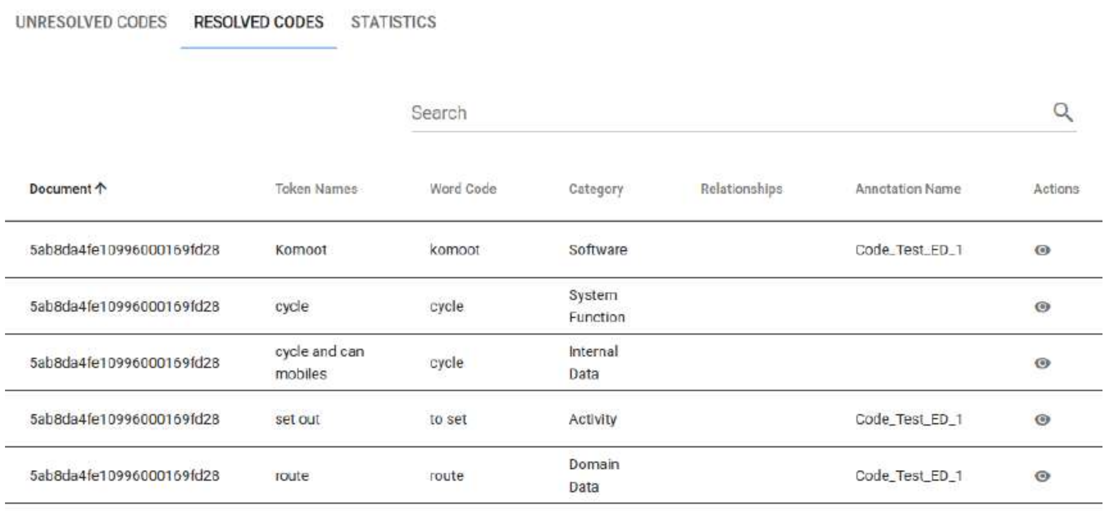
Fig. 4.12: The header of the coding results view.

In figure 4.13 a list of unresolved codes is shown. It is possible to accept or decline the codes, as well as to navigate to the occurrence in the text. The navigation is also possible from the list of resolved tokens, which can be seen in figure 4.14. Lastly, the code statistics including the kappa values as well as the possibility to update the kappa values are shown in figure 4.15.



Document ↑	Token Names	Word Code	Category	Relationships	Annotation Name	Actions
5ab8da4fe10996000169fd28	cycle route	cycle	Domain Data		Code_Test_ED_1	✓ ✕ 👁
5ab8da4fe10996000169fd28	route planning	route	Activity		Code_Test_JW_1	✓ ✕ 👁
5ab8da4fe10996000169fd28	planning	planning	Activity		Code_Test_ED_1	✓ ✕ 👁

Fig. 4.13: The tab containing the unresolved codes.

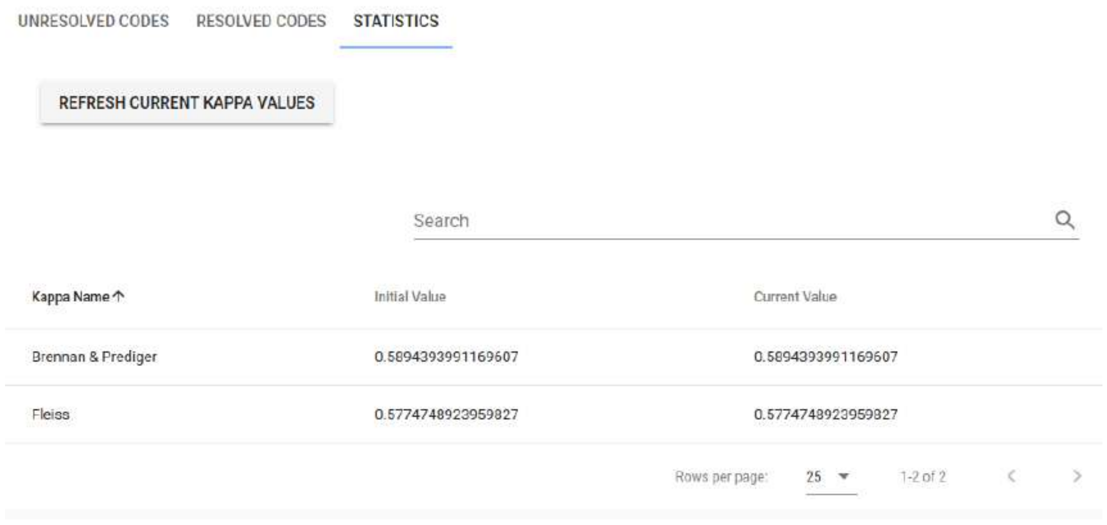

UNRESOLVED CODES | RESOLVED CODES | STATISTICS

Search

Document ↑	Token Names	Word Code	Category	Relationships	Annotation Name	Actions
5ab8da4fe10996000169fd28	Komoot	komoot	Software		Code_Test_ED_1	👁
5ab8da4fe10996000169fd28	cycle	cycle	System Function			👁
5ab8da4fe10996000169fd28	cycle and can mobiles	cycle	Internal Data			👁
5ab8da4fe10996000169fd28	set out	to set	Activity		Code_Test_ED_1	👁
5ab8da4fe10996000169fd28	route	route	Domain Data		Code_Test_ED_1	👁

Fig. 4.14: The tab containing the resolved codes.

The realization of this View contains the functions to download one table, download all tables and to export as annotation just as the mock-up in section 4.1.7, figure 4.6. However, the content of the tabs have changed a bit, because sorting by word codes, categories and relationships did not seem fitting for the tasks the user wants to achieve. Sorting by resolved and unresolved codes covers the main task, which is resolving disagreements.


UNRESOLVED CODES | RESOLVED CODES | STATISTICS

REFRESH CURRENT KAPPA VALUES

Search

Kappa Name ↑	Initial Value	Current Value
Brennan & Prediger	0.5894393991169607	0.5894393991169607
Fleiss	0.5774748923959827	0.5774748923959827

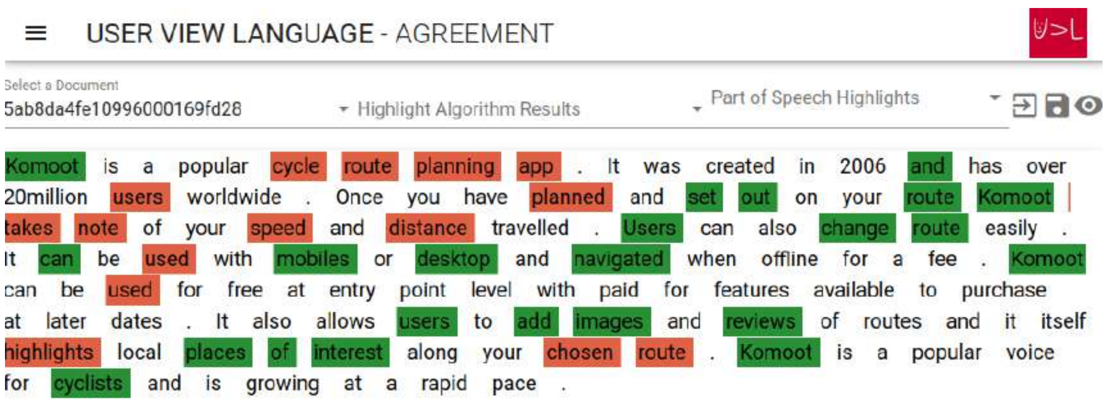
Rows per page: 25 ▾ 1-2 of 2 ‹ ›

Fig. 4.15: The tab containing the agreement statistics.

One important design decision made in this view is the update of the kappa values. It was decided that current kappa values are only updated, if the button "Refresh current kappa values" is pressed. An alternative would have been to update kappa values after every save, however, this can happen very often, as Feed.UVL automatically saves the agreement every 120 seconds. As the calculation of kappa values is computationally expensive, and the current kappa values are rarely needed, updating them only when necessary is sufficient.

Agreement Editor View

The *Agreement Editor View* realizes WS10. As can be seen in figure 4.16, the editor consists of some highlight features, navigation between documents, and displaying text. All tokens that have been encoded are coloured, if there is a disagreement they are coloured red, otherwise green. The buttons for saving are also shown in the right top corner. The basic layout of this view is very similar to the mock-up in figure 4.5 in section 4.1.7.



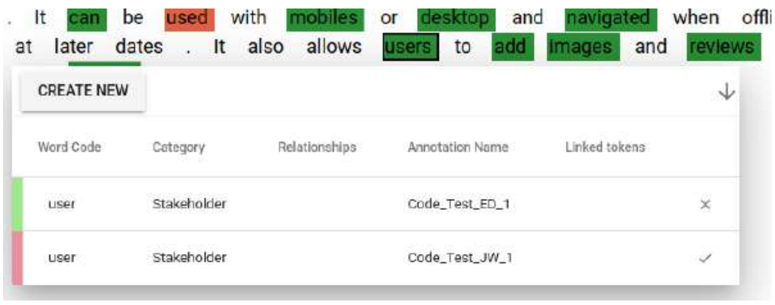
USER VIEW LANGUAGE - AGREEMENT
Select a Document
5ab8da4fe10996000169fd28
Highlight Algorithm Results
Part of Speech Highlights

Komoot is a popular cycle route planning app . It was created in 2006 and has over 20million users worldwide . Once you have planned and set out on your route Komoot takes note of your speed and distance travelled . Users can also change route easily . It can be used with mobiles or desktop and navigated when offline for a fee . Komoot can be used for free at entry point level with paid for features available to purchase at later dates . It also allows users to add images and reviews of routes and it itself highlights local places of interest along your chosen route . Komoot is a popular voice for cyclists and is growing at a rapid pace .

Fig. 4.16: The view of the agreement editor.

Agreement Editor All Codes View

The *Agreement Editor All Codes View* realizes WS10.1, and is shown when an encoded token is selected. The dialog, which can be observed in figure 4.17, consists of a list of codes with all relevant information, as well as the ability to accept or reject the codes. Accepted codes are coloured green, rejected codes are red and pending codes are not coloured at all (not displayed). When clicking on the button “Create New”, the user is navigated to the workspace WS10.2.


. It can be used with mobiles or desktop and navigated when offli
at later dates . It also allows users to add images and reviews

CREATE NEW

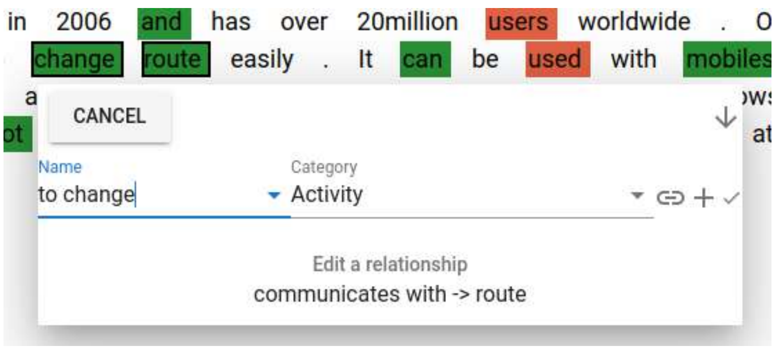
	Word Code	Category	Relationships	Annotation Name	Linked tokens	
🟩	user	Stakeholder		Code_Test_ED_1		×
🟥	user	Stakeholder		Code_Test_JW_1		✓

Fig. 4.17: The view of all codes for one token.

This View looks completely different from the mock-up as presented in section 4.1.7, figure 4.5, as there is only one list with each row containing a complete code. The implementation of a comparison by word code, category and relationship proved to be very complex because of several reasons. Firstly, the information on whether a word code, category or relationship is accepted or rejected has to be stored not by code, but by word code, category and relationship. Secondly, the export as an annotation would have been more complex, as there would have been no code structure like the one used in the annotation model, and it would not have been clear which word codes should be associated with which categories or relationships, since multiple codes could have been assigned to one token.

Agreement Editor New Code View

The *Agreement Editor New Code View* realizes WS10.2, and is displayed in figure 4.18. It is possible to add a word code, a category, as well as relationships and other tokens. The code is created when clicking on the creation button. This view did not exist in any of the mock-ups, as the first ideas never included adding new codes to tokens. The first concepts only included accepting or declining existing codes.


in 2006 and has over 20million users worldwide . O
change route easily . It can be used with mobiles

CANCEL

Name: to change
Category: Activity

Edit a relationship
communicates with -> route

Fig. 4.18: The dialog for the creation of a new code for a token.

The decision was made that a new code is only created when clicking "Add and accept", and all input is lost when clicking outside of the dialog. This decisions stems from the implementation of creating new codes in the *Annotation View* of Feed.UVL, in which codes are created automatically as soon as the dialog is opened. This implementation can lead to frustrations, because codes can be created by accident, which leads to disagreements when comparing the annotation to others. The decision was made to require intentional acceptance of a new code by clicking on a button, and to not create the code when clicking outside of the dialog. Additionally, clicking outside the dialog removes all input, because it is interpreted as no code for the tokens should be created. Otherwise it would have been frustrating if the input was persisted, and shown when a new code was created for another token.

4.3 Quality Assurance

In this section, the quality assurance is summarized. The test concept is presented in subsection 4.3.1. In subsection 4.3.2 the concept of test-driven development is introduced, and it is explained how it was included in the development process behind this thesis. The results of the static tests are provided in subsection 4.3.3. All system tests for system functions as well as some of the NFRs are outlined in subsection 4.3.4. Lastly, the most major problems and bugs found during the development or testing phases are summarized in subsection 4.3.5.

4.3.1 Test Concept

There are some major problems with testing in the Feed.UVL project. First, the frontend is relatively hard to test, because component tests are hard to implement. That means everything is tested with system tests, even methods written in Javascript. Combined with the missing development environment, the development of the frontend is hard and only system tests on the production environment can be easily used for quality assurance. Second, microservices based on *Go* are also inconvenient to test, because dependencies are missing in the local versions. Therefore the microservices are again only tested using system tests.

The first part of the test concept includes static tests for all microservices related to the implementation of the agreement functionalities. This can be useful for detecting bad smells, high complexity, or other problems with the code.

The second part consists of thorough system tests that cover all possible inputs. The system is assumed to be working correctly, when all system tests give the expected results. The test concept for system tests can be observed in table 4.16.

Table 4.16: TC: Test concept for System tests with Agreement Functionalities

TC: Test concept for System tests with Agreement Functionalities	
Test Object	All system functions that cover agreement functionalities. That means all system function related to: UT1S: Manage Agreement, UT2S: Inspect Agreement and UT3S: Resolve Disagreements
Test Coverage	At least all functional requirements and NFR2: Usability
Idea	1. All system functions are tested manually 2. Not all possible permutations are tested, but at least all classes of singular values -> minimal coverage of equivalency classes
Test Execution	Manually
Test Results	All tests are successful

4.3.2 Test-Driven Development

Test-Driven Development (TDD) is a part of agile development, and describes the development of tests before implementing features or writing code. The idea is to develop test cases and write tests before any implementation exists, and to use them as a basis for further development. Using TDD in software projects tends to lead to more correct code, because mistakes directly cause tests to fail. This in turn can lead to higher quality code, and a reduction of bugs.

A form of TDD was used during the development of the agreement features in the Feed.UVL project. If a new method or feature had to be developed, scratch files had been used. Scratch files can serve as a small, independent development environment. They can therefore be used to implement and test small features, methods or components. The approach was the following:

First, all possible test cases, especially borderline or unusual cases, are developed. Using those test cases, testdata is created and the result of processing the testdata through the feature is specified, which can be used as an expected result to compare the actual results to. Then the methods necessary to implement the feature are developed and implemented. When the results look as predicted, the developed code can be copied from the scratch file to the actual system. From here on it is very likely that the feature works correctly, because it was extensively tested.

This process has been done to develop methods with *Javascript* as well as *Go*, and serves as a good alternative for component tests at least while developing, because functionalities of methods can be tested and verified, and the rest can still be tested with system tests. The disadvantage is that they are not real tests, and as such they are not part of a test pipeline, meaning as soon as methods are changed by someone, they can become faulty without warning.

4.3.3 Static Tests

The static code analysis has been performed using *Codefactor*[6]. The number of issues in the different microservices can be observed in table 4.17. Only the *ri-visualization* microservice has issues, two of them regard unused variables, two regard styling in a css-file. The last issue is ignored, as the static code analysis recognizes a method as having a high complexity, which is correct, but justified. All other issues have been fixed.

Table 4.17: The results of the static code analysis

Microservice	Numer of Issues
uvl-agreement	0
uvl-storage-concepts	0
uvl-orchestration-concepts	0
ri-visualization	5

4.3.4 System Tests

System tests are tests which verify whether requirements are correctly implemented, which coincides with *NFR4: Functionality*, more specifically *Functional Correctness* in section 4.1.5. This includes functional requirements as well as non-functional requirements. All system tests are derived by evaluating equivalency classes for inputs of the system functions, and testing in such a manner that all classes were included in tests at least once.

[6] https://www.codefactor.io/dashboard

Navigation functions

All navigation functions were tested from all possible starting points. If the end point of the navigation included showing a list, the list was tested containing minimal results (either no results or the minimum number possible) as well as multiple results. In the case of *SF5: Show Kappa Values*, the calculation of the initial kappa value was tested as well. The system function *SF8: Navigate to Occurrence* included testing navigation to the first and second document. As part of *NFR2: Usability*, it was also assured that the navigation takes at most one click. Another metric of this NFR is tested by combining SF1 with both, SF9 and SF11. Is it assured that codes can be accepted by at most two clicks. A summary of all system tests for the navigation system functions can be seen in table 4.18.

Table 4.18: All system tests for the system functions used for the navigation between workspaces.

SF1: Show All Codes View
TCS1.1: Show all codes view with one code
TCS1.2: Show all codes view with multiple codes
SF2: Show Editor New Code View
TCS2.1: Show editor new code view
SF3: Show Code View
TCS3.1: Show Code View from Master
TCS3.2: Show Code View from Editor
SF4: Show Agreement Editor View
TCS4.1: Show Agreement Editor View from Master
TCS4.2: Show Agreement Editor View from Code View
SF5: Show Kappa Values
TCS5.1: Show Kappa Values with perfect agreement
TCS5.2: Show Kappa Values with perfect disagreement
TCS5.3: Show Kappa Values with some agreements some disagreements
SF6: Show Resolved Codes
TCS6.1: Show resolved codes, no resolved
TCS6.2: Show resolved codes, multiple resolved
SF7: Show Unresolved Codes
TCS7.1: Show unresolved codes, no unresolved
TCS7.2: Show unresolved codes, multiple unresolved

SF8: Navigate to Occurrence

TCS8.1: Navigate to occurrence from resolved to first document

TCS8.2: Navigate to occurrence from unresolved to second document

SF9: Accept code

Table 4.19 shows the system tests. Accepting codes can be done from two different workspaces, which are both tested.

Table 4.19: System tests for SF9: Accept code

TCS9.1: Accept code from Code View	
Preconditions	At least one agreement a exists, and at least one unresolved code c
Preconditions GUI	WS9: Agreement Code Results View
Test Steps	1. Click on button "Accept" for the unresolved code c
Expected Result GUI	WS9: Agreement Code Results View, the code c is accepted and no longer in the unresolved table
Expected Exception	None
Postcondition System	Code c is accepted
TCS9.2: Accept pending code from Editor	
Preconditions	At least one agreement a exists, and at least one pending code c for token t
Preconditions GUI	WS10.1: Agreement Editor All Code View
Test Steps	1. Click on button "Accept" for the pending code c
Expected Result GUI	WS10.1: Agreement Editor All Code View, and the code c is accepted and coloured green
Expected Exception	None
Postcondition System	Code c is accepted
TCS9.3: Accept declined code from Editor	
Preconditions	At least one agreement a exists, and at least one declined code c for token t
Preconditions GUI	WS10.1: Agreement Editor All Code View
Test Steps	1. Click on button "Accept" for the declined code c

Expected Result GUI	WS10.1: Agreement Editor All Code View, and the code c is accepted and coloured green
Expected Exception	None
Postcondition System	Code c is accepted

SF10: Create new code

The system tests for this system function are listed in table 4.20. It is tested whether creating a code with only a word code or a category is possible. Additionally adding one or multiple tokens, as well as adding one or multiple relationships, is tested.

Table 4.20: System tests for SF10: Create new code

TCS10.1: Create new code with one word code	
Preconditions	At least one agreement a exists. A token t is selected.
Preconditions GUI	WS10.2: Agreement Editor New Code View
Test Steps	1. Enter the word code "Some word" 2. Click "Add and Accept"
Expected Result GUI	WS10.1: Agreement Editor All Code View, and the code is accepted and coloured green
Expected Exception	"Cancel" is clicked or user clicks outside of the dialog
Postcondition System	A code is created for token t, with word code "Some word" and no category. The code is accepted.

TCS10.2: Create new code with one category	
Preconditions	At least one agreement a exists. A token t is selected.
Preconditions GUI	WS10.2: Agreement Editor New Code View
Test Steps	1. Select category "Activity" 2. Click "Add and Accept"
Expected Result GUI	WS10.1: Agreement Editor All Code View, and the code c is accepted and coloured green
Expected Exception	"Cancel" is clicked or user clicks outside of the dialog
Postcondition System	A code is created for token t, with category "Activity" and no word code. The code is accepted.

TCS10.3: Create new code with word code, category, one relationship, one added token

Preconditions	At least one agreement a exists. A token t is selected.
Preconditions GUI	WS10.2: Agreement Editor New Code View
Test Steps	1. Enter the word code "Some word" 2. Select category "Activity" 3. Click on "New Relationship" and select the adjacent token t1, select the relationship type "works with". 4. Click on "Add other tokens" and select token t2 adjacent to previous token t1. 5. Click "Add and Accept"
Expected Result GUI	WS10.1: Agreement Editor All Code View, and the code c is accepted and coloured green
Expected Exception	"Cancel" is clicked or user clicks outside of the dialog
Postcondition System	Another code is created for token t, with word code "Some word", category " Activity", a relationship "works with" to token t1 and one added token t2. The code is accepted.

TCS10.4: Create new code with word code, category, two relationships, two added tokens

Preconditions	At least one agreement a exists. A token t is selected.
Preconditions GUI	WS10.2: Agreement Editor New Code View
Test Steps	1. Enter the word code "Some word" 2. Select category "Activity" 3. Click on "New Relationship" and select adjacent token t1, select relationship type "works with". Repeat once with token t2 adjacent to t1. 4. Click on "Add other tokens" and select token t3 adjacent to token t2 . Repeat once with token t4 adjacent to token t3. 5. Click "Add and Accept"
Expected Result GUI	WS10.1: Agreement Editor All Code View, and the code c is accepted and coloured green
Expected Exception	"Cancel" is clicked or user clicks outside of the dialog
Postcondition System	Another code is created for token t, with word code "Some word", category "Activity", two relationships "works with" to tokens t1 and t2, and two added tokens t3 and t4. The code is accepted.

SF11: Reject code

Table 4.21 shows the system tests. Rejecting codes can again be done from two different workspaces, which are both tested.

Table 4.21: System tests for SF11: Reject code

TCS11.1: Reject code from Code View	
Preconditions	At least one agreement a exists, and at least one unresolved code c
Preconditions GUI	WS9: Agreement Code Results View
Test Steps	1. Click on button "Reject" for the unresolved code c
Expected Result GUI	WS9: Agreement Code Results View, the code c is rejected and no longer in the unresolved table
Expected Exception	None
Postcondition System	Code c is rejected
TCS11.2: Reject pending code from Editor	
Preconditions	At least one agreement a exists, and at least one pending code c for token t
Preconditions GUI	WS10.1: Agreement Editor All Code View
Test Steps	1. Click on button "Reject" for the pending code c
Expected Result GUI	WS10.1: Agreement Editor All Code View, and the code c is rejected and coloured red
Expected Exception	None
Postcondition System	Code c is rejected
TCS11.3: Reject declined code from Editor	
Preconditions	At least one agreement a exists, and at least one accepted code c for token t
Preconditions GUI	WS10.1: Agreement Editor All Code View
Test Steps	1. Click on button "Reject" for the accepted code c
Expected Result GUI	WS10.1: Agreement Editor All Code View, and the code c is rejected and coloured red
Expected Exception	None
Postcondition System	Code c is rejected

SF12: Create new agreement

The creation of a new agreement contains multiple inputs which have to be tested. Firstly, the user can select at least two annotations for a dataset. Here, it is tested whether the comparison of two and three annotations is working as expected. Additionally, the user can choose to automatically merge all text passages which have no disagreements, which has to be tested as well.

The output is also different depending on the codes in the annotations. The correct comparison of word codes, categories, relationships and tokens has to be included. Also, if an annotator has not assigned any code to a text passage which others have encoded, it has to be tested that there is still a disagreement. The combination of all of those concerns results is the system tests as listed in table 4.22.

Lastly, in order to quality-assure the *NFR2: Usability*, it is tested that it's not possible to create an agreement with only one annotation, or to use an agreement name that already exists.

Table 4.22: System tests for SF12: Create new agreement

TCS12.1: Create new agreement with three annotations, perfect agreement, automerge	
Preconditions	Three annotations a1, a2, a3 for dataset d exist. All of them have a perfect agreement on multiple tokens.
Preconditions GUI	WS8: Agreement Master View
Test Steps	1. Select Dataset d 2. Enter agreement name "NewAggrX" and fill "X" with a number such that the name is unique 3. Select the three annotations a1, a2, a3 4. Click "Create New Agreement"
Expected Result GUI	WS10: Agreement Editor View
Expected Exception	None
Postcondition System	The agreement is created, with no disagreements

TCS12.2: Create new agreement with three annotations, automerge	
Preconditions	Three annotations a1, a2, a3 for dataset d exist. There are at least the following tokens: one token t1 with a perfect agreement, one token t2 with two annotators agreeing and the last having no code, t3 with two annotators agreeing and the last not agreeing, t4 with all not agreeing, t5 with three same codes from one annotator

Preconditions GUI	WS8: Agreement Master View
Test Steps	1. Select Dataset d 2. Enter agreement name "NewAggrX" and fill "X" with a number such that the name is unique 3. Select the three annotations a1, a2, a3 4. Click "Create New Agreement"
Expected Result GUI	WS10: Agreement Editor View
Expected Exception	None
Postcondition System	The agreement is created, with a resolved disagreement at t1, and unresolved disagreements at t2, t3, t4, t5

TCS12.3: Create new agreement with two annotations, perfect agreement, automerge

Preconditions	Two annotations a1, a2 for dataset d exist. There is a perfect agreement, and there are the following tokens: t1 both annotations have only a word code, t2 both annotations have only a category, t3 both annotators have a word code, category, one relationship, t4 both annotators have a word code, category, two relationships, t5 both annotators have a word code, category, one added token, t6 both annotators have a word code, category, two added tokens
Preconditions GUI	WS8: Agreement Master View
Test Steps	1. Select Dataset d 2. Enter agreement name "NewAggrX" and fill "X" with a number such that the name is unique 3. Select the two annotations a1, a2 4. Click "Create New Agreement"
Expected Result GUI	WS10: Agreement Editor View
Expected Exception	None
Postcondition System	The agreement is created, with no unresolved disagreements and all codes for tokens t1 to t6 with one code accepted and rejected each

TCS12.4: Create new agreement with two annotations, perfect agreement, no automerge

Preconditions	Same as TCS12.3
Preconditions GUI	WS8: Agreement Master View

Test Steps	1. Select Dataset d 2. Enter agreement name “NewAggrX” and fill “X” with a number such that the name is unique 3. Unselect “Automatically resolve all inter-rater concurrences” 4. Select the two annotations a1, a2 5. Click “Create New Agreement”
Expected Result GUI	WS10: Agreement Editor View
Expected Exception	None
Postcondition System	The agreement is created, there are only unresolved disagreements

TCS12.5: Create new agreement with two annotations, perfect disagreement, automerge

Preconditions	Two annotations a1, a2 for dataset d exist. There is a perfect disagreement, and there are the following tokens: t1 both annotations have different a word codes, t2 both annotations have different categories, t3 both annotators have the same word code and category, but each one different relationship, t4 both annotators have the same word code, category, each one different added token
Preconditions GUI	WS8: Agreement Master View
Test Steps	1. Select Dataset d 2. Enter agreement name “NewAggrX” and fill “X” with a number such that the name is unique 3. Select the two annotations a1, a2 4. Click “Create New Agreement”
Expected Result GUI	WS10: Agreement Editor View
Expected Exception	None
Postcondition System	The agreement is created, with only unresolved disagreements and all codes for tokens t1 to t6 with only pending codes

TCS12.6: Create new agreement with two annotations, one empty annotation, automerge

Preconditions	Two annotations a1, a2 for dataset d exist. one annotator has annotated nothing, and the other has the following tokens: t1 only a word code, t2 only a category, t3 a word code, category, one relationship, t4 a word code, category, two relationships, t5 a word code, category, one added token, t6 a word code, category, two added tokens
Preconditions GUI	WS8: Agreement Master View

Test Steps	1. Select Dataset d 2. Enter agreement name "NewAggrX" and fill "X" with a number such that the name is unique 3. Select the two annotations a1, a2 4. Click "Create New Agreement"
Expected Result GUI	WS10: Agreement Editor View
Expected Exception	None
Postcondition System	The agreement is created, there are only unresolved disagreements and all codes for tokens t1 to t6 with one code pending

TCS12.7: Create new agreement with one annotation

Preconditions	An annotation a for dataset d exists.
Preconditions GUI	WS8: Agreement Master View
Test Steps	1. Select Dataset d 2. Enter agreement name "NewAggrX" and fill "X" with a number such that the name is unique 3. Select the annotation a
Expected Result GUI	WS8: Agreement Master View, button "Create New Agreement" is disabled
Expected Exception	None
Postcondition System	None

TCS12.8: Create new agreement with name that already exists

Preconditions	At least one agreement a with agreement name "MyAggr", and two annotations an1, an2 for dataset d exist.
Preconditions GUI	WS8: Agreement Master View
Test Steps	1. Select Dataset d 2. Enter agreement name "MyAggr" 3. Select the annotations an1, an2
Expected Result GUI	WS8: Agreement Master View, button "Create New Agreement" is disabled
Expected Exception	None
Postcondition System	None

SF13: Save Agreement

Saving an agreement can be done from two different workspaces, and is implemented on the GUI either as a single "Save" or as a combination of "Save & Exit". Therefore two system tests are derived, one containing the first workspace and the "Save", the other containing the other workspace and the "Save & Exit". The system tests can be found in table 4.23.

Table 4.23: System tests for SF13: Save Agreement

TCS13.1: Save Agreement from Editor and Close	
Preconditions	At least one agreement a exists
Preconditions GUI	WS10: Agreement Editor View
Test Steps	1. Click on button "Save and exit"
Expected Result GUI	WS8: Agreement Master View
Expected Exception	None
Postcondition System	Agreement a is saved

TCS13.2: Save Agreement from Code View and do not Close	
Preconditions	At least one agreement a exists
Preconditions GUI	WS9: Agreement Code Results View
Test Steps	1. Click on button "Save"
Expected Result GUI	WS9: Agreement Code Results View
Expected Exception	None
Postcondition System	Agreement a is saved

SF14: Remove Agreement

Table 4.24 contains the system test for this system function. There is only one kind of input possible, therefore there is only one system test. As part of *NFR2: Usability*, the inability to remove an agreement with less than two clicks is implicitly tested.

Table 4.24: System tests for SF14: Remove Agreement

TCS14.1: Remove Agreement	
Preconditions	At least one agreement a exists
Preconditions GUI	WS8: Agreement Master View
Test Steps	1. Click on button "Delete Agreement" for agreement a 2. Click on button "Confirm"
Expected Result GUI	WS8: Agreement Master View, agreement a is not shown
Expected Exception	Clicking on button "Cancel"
Postcondition System	Agreement a removed

SF15: Export Results As CSV

On the GUI, there are two different buttons: The first for downloading one table, the second for downloading all tables. As inputs, only empty or non-empty tables are tested. The system tests are listed in table 4.25.

Table 4.25: System tests for SF15: Export Results As CSV

TCS15.1: Export Results As CSV, all tables	
Preconditions	At least one agreement a exists, there are some disagreements, some resolved codes
Preconditions GUI	WS9: Agreement Code Results View
Test Steps	1. Click on button "Download all"
Expected Result GUI	WS9: Agreement Code Results View
Expected Exception	None
Postcondition System	Three csv-files containing tables of the three tabs are generated and ready for download
TCS15.2: Export Results As CSV, single non-empty table	
Preconditions	At least one agreement a exists, there are resolved codes
Preconditions GUI	WS9: Agreement Code Results View
Test Steps	1. Navigate to tab "Resolved" 2. Click on button "Download this table"
Expected Result GUI	WS9: Agreement Code Results View

Expected Exception	None
Postcondition System	A csv containing a non-empty table with resolved codes is generated and ready for download

TCS15.3: Export Results As CSV, single empty table	
Preconditions	At least one agreement a exists, there are no disagreements
Preconditions GUI	WS9: Agreement Code Results View
Test Steps	1. Navigate to tab “Unresolved” 2. Click on button “Download this table”
Expected Result GUI	WS9: Agreement Code Results View
Expected Exception	None
Postcondition System	A csv containing an empty table is generated and ready for download

SF16: Create new annotation from resolved agreement

As the input of the function is an agreement, there are many different equivalency classes, contained in table 4.26. The most relevant factors are whether an agreement contains no codes, and if it contains codes, whether word code, categories and relationships are correctly transformed to an annotation. It is also important to test whether codes with multiple tokens are transformed correctly. Additionally, as part of *NFR2: Usibility*, it is tested that choosing an annotation name that already exists is not possible.

Table 4.26: System tests for SF16: Create new annotation from resolved agreement

TCS16.1: Create new annotation from resolved agreement with no codes	
Preconditions	At least one agreement a exists, without any codes
Preconditions GUI	WS9: Agreement Code Results View
Test Steps	1. Enter annotation name “AnnoX”, with X a number such that the name is unique 2. Click button “Export as annotation”
Expected Result GUI	WS9: Agreement Code Results View, tip “Agreement is exported an annotation”
Expected Exception	None
Postcondition System	The annotation is created, containing no codes

TCS16.2: Create new annotation from resolved agreement with all encoding types	
Preconditions	At least one agreement a exists that is completely resolved, with exactly the following accepted codes: c1 with just a word code, c2 with just a category, c3 with a word code, category, one relationship, and one connected token, c4 with a word code, category, two different relationships and two connected tokens
Preconditions GUI	WS9: Agreement Code Results View
Test Steps	1. Enter annotation name “AnnoX”, with X a number such that the name is unique 2. Click button “Export as annotation”
Expected Result GUI	WS9: Agreement Code Results View, tip “Agreement is exported an annotation”
Expected Exception	None
Postcondition System	The annotation is created, containing exactly c1, c2, c3, c4

TCS16.3: Create new annotation from resolved agreement with an already existing name	
Preconditions	At least one agreement a that is completely resolved and an annotation with name “MyAnno” exists
Preconditions GUI	WS9: Agreement Code Results View
Test Steps	1. Enter annotation name “MyAnno”
Expected Result GUI	WS9: Agreement Code Results View, the button “Export as annotation” is disabled
Expected Exception	None
Postcondition System	None

SF17: Update Kappa Values

This system function describes the calculation of the current kappa. The system tests can be found in table 4.27, and describe the expected value of the kappas.

Table 4.27: System tests for SF17: Update Kappa Values

TCS17.1: Update Kappa Values with no change	
Preconditions	At least one agreement a exists, no changes have been made after creation

Preconditions GUI	WS9: Agreement Code Results View
Test Steps	1. Navigate to tab “Statistics” 2. Click on “Refresh Current Kappa Values”
Expected Result GUI	WS9: Agreement Code Results View, with current kappa values unchanged
Expected Exception	None
Postcondition System	None

TCS17.2: Update Kappa Values with resolved agreement	
Preconditions	At least one agreement a exists, all disagreements have been resolved
Preconditions GUI	WS9: Agreement Code Results View
Test Steps	1. Navigate to tab “Statistics” 2. Click on “Refresh Current Kappa Values”
Expected Result GUI	WS9: Agreement Code Results View, with current kappa values equal to 1
Expected Exception	None
Postcondition System	Current kappa values are set to 1

TCS17.3: Update Kappa Values with one code resolved	
Preconditions	At least one agreement a exists, one code has been resolved
Preconditions GUI	WS9: Agreement Code Results View
Test Steps	1. Navigate to tab “Statistics” 2. Click on “Refresh Current Kappa Values”
Expected Result GUI	WS9: Agreement Code Results View, with current kappa values higher than before
Expected Exception	None
Postcondition System	Current kappa values are recalculated

4.3.5 Major Bugs and Issues

During the implementation of the project, some major issues occurred and slowed down the development process immensely.

It is possible to accept multiple codes for any token. In general, there should be the possibility to accept multiple codes for one token, because a token could be assigned multiple categories. This leads to a problem with the calculation of kappa values, as one code could have multiple different encodings, and therefore the set of codes for one token has to be taken into consideration during the calculation.

The missing development environment. There are many irrelevant commits on the git branches, making tracking of the features hard. Also, developing new features takes a longer time because of the high build durations in the production environment.

The uvl-annotation container was offline because of a problem with Jenkins. During the implementation period, the uvl-annotation container was offline for some time. As the agreement functionalities depend on a running uvl-annotation microservice, it was not possible to test some agreement functionalities during that time. The problem stemmed from the deployment of another microservice using *Jenkins*, and only happened because some deployment configurations had been copied and not correctly changed.

Additionally, during the testing phase of the project and especially while executing the system tests, some bugs were found and fixed. The documentation of the bugs as well as their solutions are detailed in table 4.28.

Table 4.28: The bugs found while executing system tests

B: When creating a new code, relationships cannot be added	
Description	When creating a new code for a token, the button for the creation of new relationships is always disabled.
Steps to Reproduce	1. Create any agreement with at least one encoded token. 2. Select that token, click on "Create new". 3. Select a category that supports relationships (e.g. Activity).
Solution	The button for the creation of relationships did not recognize whether a category supports relationships. This was added.
B: Agreement cannot be exported as annotation, if "%" is in its name	
Description	If the agreement name contains some special character, such as "%", the agreement cannot be exported as an annotation.
Steps to Reproduce	1. Create any agreement with "%" in its name. 2. Resolve the agreement. 3. Click on "Export as annotation".
Solution	The agreement name is a query parameter in some of the communication between the microservices. Encoding the agreement name when used as a query parameter is sufficient.

B: Text Passages have no colour directly after creating agreement	
Description	Directly after creating an agreement, there is no colour shown for the tokens. When the agreement is selected again after saving and exiting, the colours are shown.
Steps to Reproduce	1. Create a new agreement with at least one code.
Solution	There was a problem with the page for AgreementAlternatives. If no relationships exist, the index is set to null, which means the page cannot be correctly initialized. Solution is setting the index to 0 if no relationship exists.

B: Agreement cannot be created when an annotation is empty	
Description	A new agreement cannot be created when at least one of the selected annotations is completely empty.
Steps to Reproduce	1. Select a dataset and annotations in create agreement dialog. One of the annotations should not contain any codes. 2. Click on create.
Solution	Kappa values are not calculated correctly, and are set to NaN. In this case, set kappa values to 0.

B: Export Agreement as annotation not working	
Description	The export is not working at all, no data is produced.
Steps to Reproduce	1. Create any agreement. 2. Resolve agreement. 3. Click on “Export as annotation”.
Solution	Agreement name is not decoded correctly when containing whitespaces. Addition of correct decoding of the query parameter “agreementName” solved the problem.

B: Relationships can be empty or null	
Description	A relationship of an annotation can be empty or null, when it is deleted.
Steps to Reproduce	1. Create an annotation. 2. Add any relationship to a token. 3. Remove that relationship. 4. Create agreement with that annotation.
Solution	This stems from a bug with the annotations. When creating an agreement, remove all relationships which are null or empty.

4.4 Usability Evaluation

In this section the implementation of the inter-rater agreement is evaluated. The execution on an informal usability test is documented in subsection 4.4.1. In subsection 4.4.2 an evaluation of the features based on the results of the usability test is conducted, and possible improvements are outlined.

4.4.1 Usability Test

For the evaluation of this implementation, an informal usability test was conducted by two master students of applied computer science. One of the students had developed the agreement tool, the other had never worked with the tool before. A dataset was provided, and the goal was to completely annotate the dataset using comparisons of the annotations. The annotation process consisted of stages, which would repeat until all documents were annotated and compared:

1. Both annotators annotate a certain amount of documents. The amount in the first loop was 3 documents, and 10%, 30%, 50%, 100% of the documents in later loops.
2. In each step an agreement is created with both annotations contained.
3. All disagreements are resolved by one person at a time. This includes accepting and rejecting codes, as well as creating new codes.
4. Two annotations are created from the resolved agreement.
5. The new annotations are used for subsequent loops.

The creation of new codes works a bit differently than the implementation in the annotation tool, so some difficulties were to be expected. One of the more significant differences is the design decision that clicking outside the dialog causes the new code not to be saved, so first time using this feature all input was lost.

4.4.2 Evaluation of Usability

During the usability test, some features proved to either be very convenient or work well. Firstly, the feature to automatically resolve all tokens without disagreements was very convenient. It reduced the amount of work immensely, as the same code would have had to be compared again and again, while the work on exported annotations continues. It would have been very frustrating to resolve the codes every single time, especially because it is clear that there cannot be any disagreements. Therefore, this feature is vital for comparing a large amount of data.

Secondly, the possibility to create new annotations from resolved agreements proved to be essential for a longer annotation process with multiple annotation-comparison loops. When creating a new annotation, all comparisons are not lost. Step-by-step a complete annotation, consisting of accepted codes from all annotations, can be established, while the annotators develop a better understanding of the categories and how to apply those to the dataset.

Lastly, it was convenient to be able to navigate to occurrences from the list of unresolved codes. When saving and exiting an agreement and in order to continue working at another time, finding the last document someone has worked on is very helpful. Also, when dividing work between multiple users, it is convenient to see the remaining documents, and when one annotator is unsure about the correct code, some disagreements can be left for discussion with another annotator. The list of unresolved codes can be used for finding those codes.

On the other hand, some design decisions and features posed some problems or caused frustrations. Firstly, the table containing all resolved codes was not used at all, and therefore did not have much of a purpose, as the only action possible from this view is the navigation to the occurrence in the text. This view could be removed completely, as it is at most barely needed, and the removal could improve the performance of the agreement tool.

Another frustration was caused by the *WS10.1: Agreement Editor All Code View*, which shows a list of all codes for any selected token. If there exist two codes for sets of overlapping tokens, all codes which are part of the disagreement are only shown when selecting tokens which are part of both codes. For example, suppose there is a sentence “The audio file contains no data”, and there is one code *c1* for tokens “audio file”, and another code *c2* for the token “file”. This is a disagreement, because the tokens are different, but at the same time they are overlapping. When selecting “audio”, only the code *c1* is shown, because that is the only code assigned to the first token. Only when selecting “file”, both codes are shown, and all codes of the disagreement have the possibility to be resolved simultaneously.

This can be frustrating, because users have to search for all codes for a set of tokens, and if more than two codes are involved it may not even be possible that all codes are shown simultaneously. For example, if there was another code *c3* for this example above, and this code is for token “audio”, there exists no view that shows all codes for this disagreement, as it is interpreted as two separate disagreements. Users may have to accept or decline one of the codes first, in order to see which tokens still have disagreements, which can be frustrating.

One design decision that caused frustrations is the implementation of the *WS10.2: Agreement Editor New Code View*. The dialog can be closed by clicking outside, and all previous input is removed. This was intentionally designed this way, because a new code should only be created when explicitly creating the code by clicking a button. Nevertheless, it can be frustrating for users, as a misclick can cause all input to be lost. Additionally, sometimes it was simply forgotten to accept the code by clicking the button, and all input has to be entered again.

During the usability test, one possible improvement of a feature was proposed. Before, only pending codes could be accepted or declined. The idea was to add the possibility of being able to accept and decline all codes. In practice, misclicks are possible, and the option to reverse a choice is very convenient and necessary. This improvement was implemented.

As the usability test was only informal, users were not asked to give feedback. Therefore, this evaluation is also only informal, and contains just a subjective discussion of the most convenient and most frustrating features. In the future, a formal evaluation could be conducted, by using questionnaires, interviews or surveys.

5 Classifier

In this chapter, the deep-learning-based classifier for the automatic classification of the forum data is outlined. In section 5.1, the dataset for the training of the classifier is presented, the annotation process is summarized and some statistics about the annotated training dataset are included. The implementation of the classifier is provided in section 5.2, including pre-processing, the Bi-LSTM model and the training phase. Lastly, in section 5.3 the classifier is evaluated using an annotated test dataset.

5.1 Dataset

In this section, the training dataset for the classifier is presented. In subsection 5.1.1, the extraction and form of the forum dataset is provided. The annotation process of the dataset is summarized in subsection 5.1.2, and the main issues during the comparison of the annotations are discussed. Lastly in subsection 5.1.3, the results of the annotation are provided.

5.1.1 Data

The forum used for the extraction of data is *Reddit*[1], an online forum in which users can create posts, which other users can comment on. Users can also comment on other comments, giving them a hierarchical structure. The site is divided into communities, also called *Subreddits*, which consist of posts regarding different topics, some of which are software-related. For example, a subreddit called *java* will contain posts and questions about the programming language *Java*.

In Feed.UVL there is a *Reddit-Crawler* implemented, with which posts can be crawled and a dataset can be established. If the names of subreddits are provided, the crawler will crawl posts from these subreddits that were published within a specified time frame,

[1]https://www.reddit.com/

and convert them into datasets that can be used in Feed.UVL. A single document of this dataset contains the title, the content of the post and all comments up to any desired hierarchical level. The title, content and comments are separated by the separator "###", an example of which is the following:

> *Is it possible to mute all other tabs except for the active tab? ###I can't find any settings or extensions that do this ### Is what I use which has > Mute all except current one - Alt+Shift+N ###Try right click on tab and select Mute site.*

Two comments existed for this post, both of them are separated with the "###"-separator. All posts used in the dataset are taken from three different subreddits, which are *Chrome*, a commonly known browser, *Komoot*, an app for planning and navigating outdoor activities like hiking or biking, and *VLC*, a player for videos and movies. The total number of posts is 98, of which 40 are about *Chrome*, 20 about *Komoot* and 40 about *VLC*. The *Reddit*-crawler removes all URLs and emojis within the posts, and can skip posts which are too short or contain words on a pre-configured blacklist.

5.1.2 Annotation Process

Various functionalities of Feed.UVL were used for the annotation process: the *Reddit-Crawler* was used to crawl and upload the un-annotated training dataset, the *Annotator* was used for the annotation of the data, and the *Agreement*-tool was used to compare the annotations and create a new encoding, which in turn is used to train the Bi-LSTM classifier. The annotation was performed in multiple stages, in which the dataset was encoded with TORE-categories, as demonstrated in section 2.3.

The stages consisted of an annotation and a subsequent comparison phase. During the annotation phase, the work of all annotators is independent, with only the description of the TORE-categories serving as guides. Afterwards in the comparison phase, annotations are compared and disagreements are resolved such that all tokens have at most one code. The result of the comparison phase serves as a starting point for the annotation phase of the next stage. For each stage, an increasing amount of documents is annotated and compared, until all documents are fully encoded with the codes accepted during the comparison.

In this thesis, two annotators, both master students of applied computer science, were asked to annotate the data. The whole process took about 20 hours per person, over the span of some weeks. The stages consisted of the annotation of three documents, then of 10%, 30%, 50% and 100% of the data, the kappa values of which can be seen in table 5.1.

Table 5.1: The kappa values of the annotation stages.

Kappa of	3 Docs	10%	30%	50%	100%
Brennan & Prediger	0.4824	0.5244	0.7218	0.5793	0.7036
Fleiss	0.4700	0.5090	0.7180	0.5670	0.6994

The first two stages still contained many disagreements, mainly due to different annotation conventions between the annotators and due to the fact that not as many resolved codes from previous stages were included. Some of those problems were resolved by discussions between the annotators, but due to the complexity of the TORE-categories and the ambiguity of the context in which tokens occur, the kappa values remained relatively low through the different stages, with the exception of the third stage. The most common issues encountered during the comparison are the following:

- **It is unclear, whether a text passage is an *Activity* or an *Interaction*.** Suppose the documents contain a post about *Komoot*, an application used for route planning and navigation. The text passage "planning a route" is usually annotated as an *Activity*, because is can be done without the software, and it is one of the main activities of a user. Nevertheless, in some cases it can also be an *Interaction*, especially if the context implies the use of *Komoot* while planning a route.

- **The amount of tokens that have to be included in a code is ambivalent.** All annotators were to reduce the amount of tokens for each code as far as possible, which led to some disagreements regarding not the category, but the amount of tokens. For example, the text passages "work" and "work offline" can both be activities; in most cases only "work" can be annotated as it is a sufficient description of the activity, while sometimes in context it is important to annotate "work offline". Additionally, there are text passages for which the meaning of a word could be changed. Consider the text passages "get" and "get up"; the meaning of both verbs is completely different, and the word "up" should be included in the code.

- **It is unclear whether to assign the category *Workspace* or *Interaction Data*.** Sometimes it is unclear whether a user means a UI-element or the data behind it when talking about an interaction. In documents which contain posts about *VLC*, a “song” can either refer to the song as data, such as “playing a song”, or to a UI-element representing that song, such as an icon.

5.1.3 Annotation Results

The fully annotated training dataset consists of 98 documents, of which 40 are about Chrome, 20 about Komoot and 38 about VLC. The total amount of tokens is 13,775 divided into 774 sentences, and the number of codes is 2,036. The distribution of the categories over all tokens can be observed in table 5.2.

Table 5.2: The distribution of categories over all tokens.

Category	Number of Occurrences	Number of Appearances	Percentage of Annotations
Activity	85	38	3.48%
Domain Data	280	61	11.47%
Goals	0	0	0.00%
Interaction	572	91	23.42%
Interaction Data	443	82	18.14%
Internal Action	53	29	2.17%
Internal Data	58	23	2.38%
Software	567	87	23.22%
Stakeholder	33	20	1.35%
System Function	5	3	0.20%
Task	1	1	0.04%
Workspace	345	47	14.13%
None	11,333		

The “Number of Occurrences” refers to the occurrences of the code in the text, while the “Number of Appearances” reference the number of documents in which the code appears at least once. An additional category, *None*, is introduced and assigned to all tokens which have not been assigned any code. All in all, the vast majority of the tokens, about 82%, is not assigned any code. The last column of the table shows the percentage of categories used in the code when excluding all tokens that are not annotated. It can be seen that some categories are assigned very rarely or not at all, such as *Goals*, *System Function* and *Task*. Only three categories, *Interaction*, *Interaction Data* and *Software*, appear in over 85% of the documents.

5.2 Implementation

In this section the implementation of the classification algorithm is presented. In subsection 5.2.1, the coarse requirement is presented, and the deduced requirements are briefly mentioned. The pre-processing and the word embedding pipelines of the datasets are provided in subsection 5.2.2 and subsection 5.2.3, respectively. In subsection 5.2.4 the implementation of the Bi-LSTM model is provided, while the configuration and training of the model is presented in subsection 5.2.5.

5.2.1 Requirements

The implementation of the classification algorithm is based on the following coarse requirement:

R4 Algorithm for the TORE classification: Feed.UVL offers an algorithm for the automatic classification of natural language datasets into TORE-categories through the use of deep learning. A corresponding algorithm is found through the literature review and is implemented as agreed upon with the advisor. The newly implemented algorithm uses the existing views and functionalities for the management of classification algorithms in Feed.UVL. This applies to the management of the algorithm itself, as well as the management of the results of the automatic classification.

All requirements relevant to classifiers already exist in the Feed.UVL project, and are therefore not documented again. The paper used as a basis for the implementation of the algorithm is Li et al. [14], which is the result of the literature review in chapter 3.

5.2.2 Pre-processing

The training dataset has the format of an annotation as implemented in Feed.UVL, and has to be transformed in order to be used as an input for the deep learning model. The annotation object is already tokenized, and contains a list of tokens, which consist of the fields *Index* containing the position of the token in the dataset, *Word* containing the word, *Lemma* containing the lemmatized word, *POS* containing the POS tag, and the additional fields *NumberNameCodes* and *NumToreCodes*, which are used for the optimization of annotations in Feed.UVL. The lemmas are chosen as representatives of the words, as all of them are transformed into word embeddings in the next step, in section 5.2.3. The index is used in order to find the correct position of a word in the dataset after the prediction of tags using the model. All other fields are ignored, as they are irrelevant for the classification algorithm.

Additionally, the annotation contains a list of codes, which carries information on the tokens the codes apply to, as well as the TORE categories. The relevant fields of the list of tokens are combined with the relevant fields of the list of codes, forming a list containing all relevant information for all tokens, henceforth referred to as the modified list of all tokens. The training dataset also has a list of all documents, in which the start- and end-index for all documents is provided. The modified list of all tokens is split into documents, and further split into sentences. The separators chosen for the datasets include ".", "?", "!" and "###", which is the separator of all forum data parsed by the *Reddit*-crawler in Feed.UVL.

At first, stop-word removal and the removal of punctuation in the dataset were implemented, since those pre-processing methods tend to lead to better results when using deep learning algorithms for the classification of natural language datasets. However, the results during the configuration of the training pipeline showed that the accuracy increased when not using these data cleaning methods. Therefore the methods were removed.

It has to be mentioned that datasets used for the prediction of categories as implemented in Feed.UVL have a different format compared to the training dataset, and are generally not tokenized. However, all datasets can be tokenized using *NLTK*[2] with the same configurations as done in the *Annotator* mircoservice in Feed.UVL, and parsed into the structure as used in the modified list of tokens. Therefore a similar pre-processing pipeline can be applied.

In order to create a map containing all available categories, the categories are directly taken from Feed.UVL, and another category called *None* is added for all tokens which do not have a category.

5.2.3 Word Embedding

Word embeddings are used in NLP to create semantically meaningful representations of words, in the form of word vectors. Similar words are supposed to have similar word vectors, which would give them similar outputs when evaluated by a machine learning model. Word embedding models can be implemented and trained from the ground up,

[2]https://www.nltk.org/

but that requires large amounts of data and a large vocabulary, which are both not present in this thesis. However, a pre-trained word embedding model can be used for the creation of word vectors, as long as it is similar enough to the texts in the training dataset.

Li et al. [14] create a word embedding model with a dimension of 128 using the *Gensim library*[3], which is why one of the pre-trained models of this library is chosen for this thesis. The list of all pre-trained models provided by *Gensim* can be seen in table 5.3. The most fitting pre-trained model in terms of the words which might occur in the vocabulary is based on *Twitter*, and the closest dimension of the pre-trained models is 100. Therefore the model for the creation of word embeddings in this thesis is *glove-twitter-100*[4]. The kind of speech used on *Twitter* should be similar enough to the language used in online forums, and the model should contain most words and fitting similarities as the words used in the forum data, although it might lack the software relevancy that the crawled *Reddit* dataset contains. The dataset the word embedding model is based on contains 2B tweets, 27B tokens and a vocabulary of 1.2M words.

Table 5.3: All pre-trained models of the Gensim library.

Model	Source of Dataset	Dimension
fasttext-wiki-news-subwords-300	Wikipedia	300
conceptnet-numberbatch-17-06-300	Multiple Different	300
word2vec-ruscorpora-300	Russian National Corpus	300
word2vec-google-news-300	Google News	300
glove-wiki-gigaword-50	Wikipedia	50
glove-wiki-gigaword-100	Wikipedia	100
glove-wiki-gigaword-200	Wikipedia	200
glove-wiki-gigaword-300	Wikipedia	300
glove-twitter-25	Twitter	25
glove-twitter-50	Twitter	50
glove-twitter-100	Twitter	100
glove-twitter-200	Twitter	200

In order to transform the extracted sentences from the pre-processing pipeline into the correct input for the Bi-LSTM model, all sentences are padded or truncated to a specified sentence length, which through various optimizations is set to 80, and word vectors are generated for all sentences through the look-up of individual words in the word embedding model. If the word is not found in the model, all dimensions of the

[3]https://radimrehurek.com/gensim/models/word2vec.html
[4]https://nlp.stanford.edu/projects/glove/

word vector are set to 0. After this step, all sentences have the same size, which is 80x100. All the tags from the previous step are also padded or truncated to the correct size, and another category, "_ ", is added to signal the padding. All tags are mapped to numerical values and then one-hot-encoded, in order to use them in the model.

5.2.4 Bi-LSTM Model

For the creation of the Bi-LSTM model, the *Tensorflow-Keras*[5] library is used. The model consists of an input layer, two hidden bidirectional layers with a dropout layer in between, and an output layer. The shapes of the layers can be observed in table 5.4. "None" is a value used by *Keras* which represents a value of a dimension that is not fixed, and is replaced by the batch size used for the training data. The input layer has the shape of the batch size, the length of the sentence which is set to 80, and the dimension of the word embeddings, which is 100. There is no embedding layer used in this model, because the word embeddings are generated in a previous step.

Table 5.4: The input and output shapes of the Bi-LSTM layers.

Layer	Input Shape	Output Shape
Input	(None, 80, 100)	(None, 80, 100)
Bidirectional Hidden	(None, 80, 100)	(None, 80, 200)
Dropout	(None, 80, 200)	(None, 80, 200)
Bidirectional Hidden	(None, 80, 200)	(None, 80, 100)
Output Layer	(None, 80, 100)	(None, 80, 14)

The hidden layers are used to reduce the dimensions of the LSTM layers, but as they are bidirectional, the dimension of the output shapes are doubled. The first hidden layer consists of two LSTM-layers with a dimension of 100 each, the second hidden layer consists of two LSTM-layers with a dimension of 50 each. This reduction of dimensions is consistent with the idea of Li et al. [14]. The dropout layer between the two hidden layers is used to reduce overfitting of the model, and the output layer reduces the dimensions to the correct size, which is a one-hot-encoding of the predicted tags for all words.

The model is configured to use an *Adam* optimizer with the default configuration, which includes a learning rate of 0.001; *Categorical Cross-Entropy* as a loss function, as the problem is a multi-class problem; and the metric *Accuracy*.

[5]https://www.tensorflow.org/api_docs/python/tf/keras

5.2.5 Training

The training phase of the model is executed on the annotated training dataset as described in section 5.1. The raw data has the structure of an annotation, and a modified list of tokens is generated as described in subsection 5.2.2. The modified tokens are then grouped into documents, and sentences are extracted further. An example of the relevant fields of tokens that form a sentence are the following:

```
[{‘index’: 0, ‘name’: ‘Chrome’, ‘lemma’: ‘chrome’},
 {‘index’: 1, ‘name’: ‘double’, ‘lemma’: ‘double’},
 {‘index’: 2, ‘name’: ‘click’, ‘lemma’: ‘click’},
 {‘index’: 3, ‘name’: ‘bug’, ‘lemma’: ‘bug’},
 {‘index’: 4, ‘name’: ‘issue’, ‘lemma’: ‘issue’},
 {‘index’: 5, ‘name’: ‘still’, ‘lemma’: ‘still’},
 {‘index’: 6, ‘name’: ‘a’, ‘lemma’: ‘a’},
 {‘index’: 7, ‘name’: ‘thing’, ‘lemma’: ‘thing’},
 {‘index’: 8, ‘name’: ‘in’, ‘lemma’: ‘in’},
 {‘index’: 9, ‘name’: ‘2022’, ‘lemma’: ‘2022’},
 {‘index’: 10, ‘name’: ‘?’, ‘lemma’: ‘?’},...]
```

These tokens are combined with the categories of the codes if encoded, the rest is provided with the category *None*, resulting in the following representation of a sentence:

```
[(‘chrome’, ‘Software’, 0),
 (‘double’, ‘Interaction’, 1),
 (‘click’, ‘Interaction’, 2),
 (‘bug’, ‘None’, 3),
 (‘issue’, ‘None’, 4),
 (‘still’, ‘None’, 5),
 (‘a’, ‘None’, 6),
 (‘thing’, ‘None’, 7),
 (‘in’, ‘None’, 8),
 (‘2022’, ‘None’, 9)]
```

The sentences are further padded or truncated to a fixed length, and the pre-trained word embedding model presented in section 5.2.3 is used to generate word embeddings with a dimension of 100, resulting in a representation of 80 word vectors of size 100 for each sentence. The tags are mapped to numbers and one-hot-encoded, resulting in a vector of length 14 for all words. The word embeddings and the encoded tags are used as an input for the Bi-LSTM model as presented in section 5.2.4. The model is fitted using the word and tag vectors, with a batch size of 32 and 50 training epochs. Additionally, there is a validation split of 0.1 from the training data. The accuracy and loss of the training with this configuration can be observed in figure 5.1 and figure 5.2, respectively. The accuracy of the model for the validation set stays at about 96%, while the loss for the validation stays under 0.2. The training of the model with this configuration took about half an hour with an Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz processor.

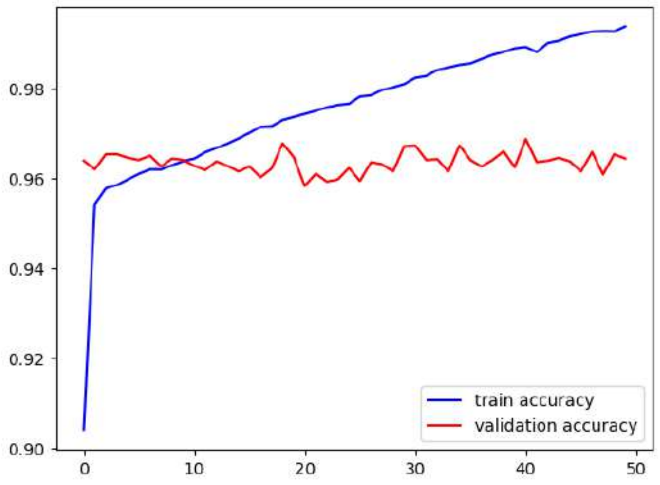


Fig. 5.1: The accuracy over all epochs.

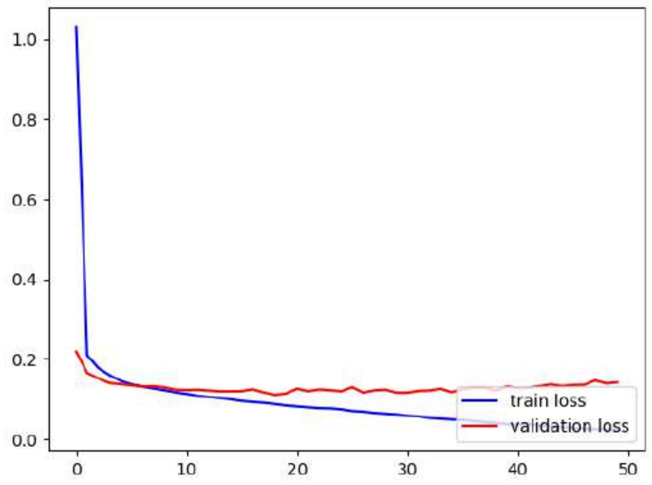


Fig. 5.2: The loss over all epochs.

An overview of all the classifier pipeline can be observed in figure 5.3. All green parts represent steps exclusively used in the training pipeline, the rest are also using for the prediction of tags. The data for the prediction is not tokenized, which is why a tokenizer has to be used first. The pre-processing, sentence extraction and word embedding steps works the same for both, the prediction and training phase. During the training phase, all categories have to be translated into numerical tags and one-hot-encoded. Lastly, during training the output of the Bi-LSTM model is used to improve the model, while the output during prediction are the predicted tags.

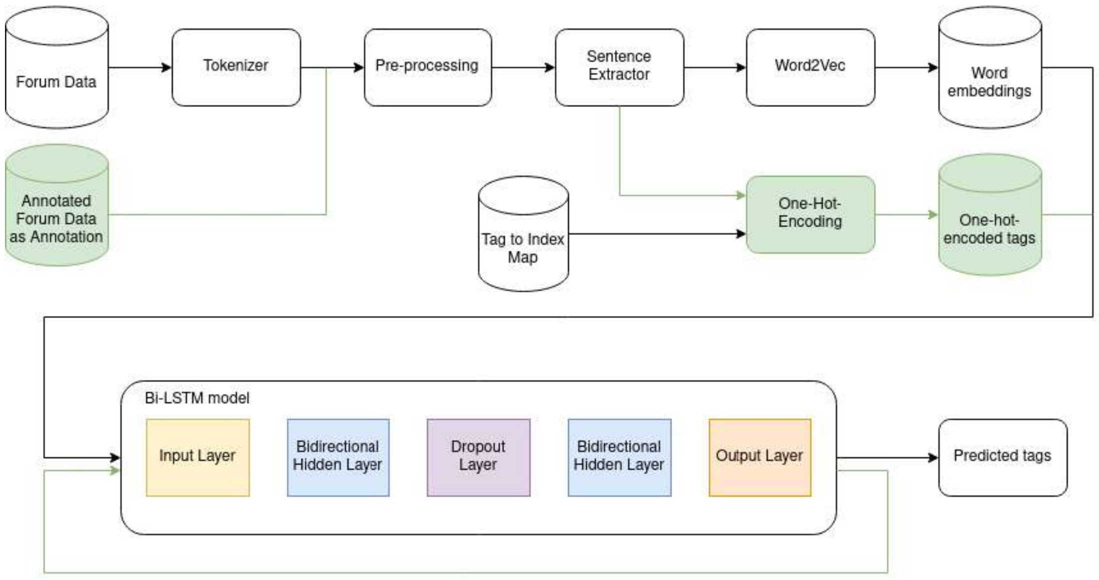


Fig. 5.3: Overview of the Bi-LSTM classifier.

In order to optimize all parameters, some additional configurations were executed as well. Firstly, it was tested whether the data cleaning methods, the stop word removal and the removal of punctuation, had a positive influence on the accuracy of the test dataset, which was not the case. In fact, data cleaning reduced the accuracy on the test dataset, which is why it was completely removed. Secondly, the length of the sentences and the number of epochs were changed and tested, resulting in the configuration described above. The accuracies of the models when evaluated on the test dataset as presented in the following section 5.3.1 can be seen in table 5.5. The accuracy was calculated on all predictions, as well as only predictions on tokens which are not assigned the tag *None*. The dimensions of the hidden layers of the model were also changed and tested, which is not included here.

Table 5.5: The accuracies of the different configurations.

Sentence Length	Number of Epochs	Is filtered?	Accuracy with *None* (in %)	Accuracy without *None* (in %)
40	25	Yes	70.48	29.95
40	50	Yes	72.85	36.65
60	25	Yes	70.53	33.72
60	40	Yes	71.42	35.91
60	50	Yes	72.82	37.95
60	70	Yes	71.08	34.68
80	25	Yes	69.46	27.37
80	25	No	83.96	34.83
80	50	Yes	71.19	35.38
80	50	No	85.26	39.95

5.3 Evaluation

This section contains the evaluation of the deep learning classifier. In subsection 5.3.1, the test dataset is presented. The results of the evaluation of the classifier on the test dataset are provided in subsection 5.3.2. In subsection 5.3.3 the most important results are summarized and discussed.

5.3.1 Test Dataset

The test dataset was created and annotated similarly to section 5.1, but the annotation process was not divided into several phases. The dataset consists of 12 documents, containing four documents for each, Chrome, Komoot and VLC. The total amount of tokens is 1,430, the number of sentences is 73, and the number of codes is 222. Table 5.6 shows the distribution of categories in the test dataset. The categories *Goals*, *Stakeholder*, *System Function* and *Task* have no occurrences, *Activity*, *Internal Action* and *Internal Data* have very few, with a combined percentage of less than 5%. Only three categories, *Interaction*, *Interaction Data* and *Software* appear at least once in all or almost all documents.

Table 5.6: The distribution of categories over all tokens for the test dataset.

Category	Number of Occurrences	Number of Appearances	Percentage of Annotations
Activity	5	2	2.25%
Domain Data	18	6	8.11%
Goals	0	0	0.00%
Interaction	81	12	36.49%
Interaction Data	49	11	22.07%
Internal Action	2	2	0.90%
Internal Data	2	2	0.90%
Software	37	11	16.67%
Stakeholder	0	0	0.00%
System Function	0	0	0.00%
Task	0	0	0.00%
Workspace	28	9	12.61%
None	1,157		

5.3.2 Results

The test data from the previous section 5.3.1 is used to evaluate the trained deep learning model. After the prediction of the tags and through the comparison with the true tags, the precision, recall and F1-scores is calculated for each TORE-category. The results of the prediction per category can be observed in table 5.7. The categories *Goals*, *Stakeholder*, *System Function* and *Task* are all evaluated at 1.0, as there are no occurrences of the categories in the test dataset, and the model's prediction did not contain the categories as well. Therefore, by definition, the categories were predicted perfectly, setting all metrics to 1.0. Similarly, the categories *Internal Action* and *Internal Data* are wrongly predicted, but both categories are contained in the test dataset, which leads to all metrics being 0.0.

Table 5.7: The precision, recall and F1-score of all categories, all values are in %.

Category	Precision	Recall	F1-score
Activity	12.50	20.00	15.38
Domain Data	36.36	46.15	40.67
Goals	1.0	1.0	1.0
Interaction	70.42	60.97	65.35
Interaction Data	49.09	36.48	41.86
Internal Action	0.00	0.00	0.00
Internal Data	0.00	0.00	0.00
Software	63.82	66.66	65.21
Stakeholder	1.0	1.0	1.0
System Function	1.0	1.0	1.0
Task	1.0	1.0	1.0
Workspace	77.41	64.86	70.58
None	92.30	94.11	93.20

All metrics regarding the category *Activity* are relatively bad, which could be due to the small number of occurrences in the test dataset. The low precision further points to the existence of an imbalance, and a high number of false positives. The predictions of the category *Domain Data* are better than those for *Activity*, however the precision is comparatively low. The reason behind this could be the similarity of *Domain Data* and *Interaction Data*, as in some cases the difference is only a previous or following verb describing either an activity or an interaction. The model may not be able to distinguish between the two categories, and therefore predict the category as easily. On the other side, *Interaction Data* has a relatively high precision and a relatively low recall, which means the number of false negatives is higher, and supporting the idea of the similarity between the categories.

Interaction, *Software* and *Workspace* are all predicted relatively well, with F1-scores of more than 60%. As all of them have a relatively high number of occurrences in the training as well as the test dataset, the model might have been trained better on these categories, making the prediction easier. Additionally, *Interaction* is the most dominant category containing verbs with the largest number of occurrences and appearances; *Software* generally contains a very small pool of words, such as “Chrome”, “Komoot”, “app”, “phone”, etc. The category *Workspace* also has a smaller number of words, and, as a contrast to most other categories, can have adjectives.

The high metrics of the category *None* are probably due to the large amount of data in the training set, as well as the large number of words which are completely unrelated to the encoded tokens, which gives them a completely different representation as a word embedding.

Generally, the Bi-LSTM model predicts more false negatives than false positives, as can be seen in table 5.8. This holds true for most categories in this evaluation, and gives an indication that the model predicts too few occurrences of the category. The category *Interaction Data*, for example, has 28 false positives and 47 false negatives, which could be interpreted as the model not assigning *Interaction Data* often enough. The categories *Internal Action* and *Internal Data* only have false negatives, and as only two occurrences of these categories each exist in the test data, both are completely missed. There are some categories which have slightly more false positives than false negatives, namely *Activity* and *Domain Data*. Both categories have significantly more occurrences relative to the total amount of codes in the training data than in the test data, so the model might have expected to encounter those more often. The category *None* also seems to have more false positives, pointing to the classifier assigning the category more often than others.

Table 5.8: The number of false positives and false negatives per category.

Category	False Positive	False Negative
Activity	7	4
Domain Data	21	14
Interaction	20	32
Interaction Data	28	47
Internal Action	0	2
Internal Data	0	2
Software	17	15
Workspace	7	13
None	80	60

Depending on whether the categories which are not included in the test dataset are incorporated and whether the category *None* is included as well, the overall metrics can change. An overview of the overall evaluation metrics is provided in figure 5.9. Precision and recall are relatively similar in all considerations. The overall metrics are always higher when the category *None* is included, and even higher if the categories which are not used in the test dataset are incorporated as well. The most interesting consideration is most likely the last entry in the table, as it is important to recognize that the model did not predict any occurrence of categories, which were not present in the test data. Therefore, the missing categories should be included in the consideration, while the high number of occurrences of the category *None* may distort the metrics. All in all, the accuracy of the model can be evaluated at about 85.38% when including *None*, and 52.74% when excluding it.

Table 5.9: The overall accuracy, precision, recall and F1-score, all values are in %.

Category	Accuracy	Precision	Recall	F1-score
Categories in prediction, including None	85.38	40.19	38.93	39.23
All categories, including None	85.38	57.28	56.38	56.59
Categories in prediction, excluding None	52.74	34.40	32.79	33.23
All categories, excluding None	52.74	54.59	53.47	53.78

5.3.3 Discussion

The implementation and evaluation of the deep learning classifier has several problems, starting with the training dataset. Firstly, the amount of data necessary to train a Bi-LSTM algorithm for the classification of a multi-class problem is very likely higher than what was provided as the training dataset. The amount of categories used in the TORE-classification is relatively high, while the total number of codes divided between all 12 categories is only 2,036. In the future, the number of categories could be adjusted by using TORE-levels instead of TORE-categories, which would reduce the number of categories to three. Secondly, the training data is very imbalanced, since some categories have a high amount of occurrences, while others occur either rarely or not at all. A more balanced training dataset could possibly have helped, or, if there were more data, techniques like over- or under-sampling. One category, *Goals*, did not occur at all in the dataset, and can therefore not be predicted by the model at all.

The test dataset embodies the next problem. Four of the 12 categories did not appear at all, which makes the prediction and evaluation of these categories impossible. Some have a low number of occurrences, making the dataset imbalanced. The category *Activity*, for example, has a relatively low number of five occurrences, with the words being “go”, “take”, “take”, “navigate”, “remember”. At least three of those are very general without context, making the prediction harder for the deep learning model.

The pre-processing of the dataset also has some flaws. Usually in NLP, some data cleaning is used to optimize the dataset for the training phase of the model. Popular techniques include stop word removal and the removal of punctuation, which is why those techniques were implemented at first. However, since the Bi-LSTM model uses backward and forward memory, the removal of stop words may lead to a decline of accuracy of the trained models, which is why the data cleaning was removed. Since backward and forward memory involves using words directly surrounding a word in the classification process, there are cases in which the context is completely removed. For example, the sentence "Is this still an issue with you guys?" is reduced to "still issue guy", which could be hard for a Bi-LSTM model to classify. The stop word removal could have lead to a more balanced model overall, as not as many tokens with category *None* would have been included. Some further work may be necessary for a more fitting pre-processing.

Some more problems are introduced with the use of a pre-trained word embedding model. As there was not sufficient data to train a new word embedding model on the *Reddit*-forum data, a pre-trained model had to be chosen. The model ultimately used is based on tweets from *Twitter*, which though it is based on natural language similar to the one used in online forums, is not specifically about software and may not contain all relevant words in the domain. For example, the training dataset contains 1,828 different words, which is a very small vocabulary. Of those, 209 words used in 363 occurrences could not be found in the pre-trained word embedding model at all. Most of them contain either special characters or numbers, nevertheless there are some words which were annotated in the training dataset, including word such as "double-clicking", "overwrote" and "mouse-over". As those words do not occur in the word embedding model, they are mapped to a vector containing only zeros, which has multiple consequences: Firstly, the model learns to assign categories which are not a placeholder to the zero-vector. In the prediction on the test dataset, the model assigns the placeholder-tag "_" to a word from the dataset, which should not happen. Secondly, the model learns that some zero-vectors can have categories which are not *None* or the placeholder-tag. Thirdly, as Bi-LSTM uses forward and backward memory, using zero-vectors has consequences for the surrounding words, as the input can consist of sentences, which appear to have a "hole" in the middle of them. This could make the classification of all other words harder.

Lastly, the evaluation of the metrics as done in section 5.3.2 shows the difference of the prediction metrics between the different categories. Some categories are not included in the dataset at all, meaning they cannot be predicted, while others are predicted completely wrong leading to metrics of 0.00%. The categories with a higher number of occurrences seem to be classified more accurately, while categories with lower numbers of occurrences have lower F1-scores. The overall accuracy of the model is about 52.74% when excluding the category *None*, which, again, points to the many flaws as discussed above. As of now, it is questionable whether the classifier can be used for the classification of TORE-categories in general, especially if the datasets are from different forums or on different software topics than *Chrome*, *Komoot* or *VLC*. However, the accuracy of the classifier is not much worse than the kappa values achieved by the human annotators, as documented in subsection 5.1.3. As the first attempt at using deep learning for the classification of natural language user feedback from online forums into TORE-categories, the results are promising and motivate further research.

6 Summary & Future Work

This chapter concludes the thesis by providing a short summary of all results in section 6.1, and presenting possible future work for the inter-rater agreement in section 6.2 and the classifier in section 6.3.

6.1 Summary

An inter-rater agreement tool supporting the comparison of multiple annotations is implemented in Feed.UVL. The tool lets users create and manage inter-rater agreements from multiple annotations of datasets, and supports the possibility to inspect agreements and to resolve disagreements. Once the agreement is resolved, the tool supports the further use of annotation functionalities, by creating a new annotation from the resolved agreement. The inter-rater agreement tool supports the calculation of initial and current kappa values, and reuses many functionalities and views of the already implemented annotation. The tool is quality assured through the use of extensive system tests, as well as static code tests and the use of TDD during the implementation phase. The agreement tool is evaluated on an informal usability test, to ensure the functional correctness and completeness, as well as to cover all coarse requirements as presented in subsection 4.1.1.

A deep-learning-based classifier for the classification of forum data into TORE-categories is implemented and evaluated, and reuses the existing views and functionalities for the management of classification algorithms in Feed.UVL. The classifier is trained, and the accuracy, precision, recall and F1-score are evaluated using a test dataset. The overall recall and precision of the classifier are relatively similar, with the accuracy being about 52.74%. As the evaluation metrics are generally not the best, with a precision of 54.59%, recall of 53.47% and F1-score of 53.78%, the model only presents a starting point for further research into automatic TORE classification even if it is not applicable by itself yet. Further, the evaluation itself is problematic, with questions on whether categories that do not occur in the test dataset should be taken into account. Another evaluation of the classifier should be conducted in the future, in order to find out whether the classifier can successfully classify all categories of the forum data.

6.2 Future Work: Inter-Rater Agreement

The implementation of the inter-rater agreement can be improved on multiple levels. Firstly, the possibility to undo the last step could be added. While it is possible to change the status of all codes to *Accepted* and *Declined*, it is not possible to change the status back to *Pending*. This can lead to a situation, in which a disagreement is resolved by one person by mistake, even though the person is unsure of the correct code. As soon as the code is resolved, it is hard to find the occurrence in the text, which could lead to a long search or a wrongly assigned label. The possibility to undo such a mistake and set a code back to the *Pending* status could help.

Further, solutions for the frustrations of users as mentioned in subsection 4.4.2 can be found and implemented. For example, the creation-dialog of a new code could be turned to a modal dialog, which requires users to either accept or decline the newly created code, making it impossible to loose the input by accident. It could also be implemented that for all tokens which are part of overlapping codes, the list of all codes always shows all overlaps, no matter whether the token itself is part of all the overlapping codes.

Secondly, more statistics could be added to the inter-rater agreement. In addition to the existing kappa values, more kappa values or correlation coefficients could be added. If the dataset contains documents on multiple software topics, it could also be interesting to see the kappa values for each individual topic. For example, the training dataset used for the classifier as presented in section 5.1 is based on three different kinds of software, *Chrome*, *Komoot* and *VLC*. It would be interesting to observe the values of the inter-rater agreement for all software products individually.

Lastly, in order to evaluate the inter-rater agreement tool properly, a formal usability test should be conducted, with proper preparation, documentation and evaluation. Additionally, a questionnaire or interviews could be conducted on multiple users, in order to evaluate the usability on different user groups.

6.3 Future Work: Classifier

The datasets used for training as well as testing can be improved in the future. The training dataset should contain a higher amount of documents, and include more software-related topics, in order to diversify the vocabulary learned by the Bi-LSTM model. It should also include all TORE-categories, and have more occurrences of categories, which appear only rarely. Similarly, the test dataset should contain all categories in order to evaluate the model properly. As of now, the evaluation of the model has many flaws as presented in subsection 5.3.3, which makes a complete reevaluation of the model necessary.

A word embedding model for software-related categories in online forums could be developed by crawling large amounts of data and training a word embedding model. This could improve the calculation of similarities between words used in software-related contexts, and lead to better word representations and fewer zero-vectors for the datasets. The reduction of words which cannot be found in word embedding models can be especially significant, as the Bi-LSTM has forward and backward memory. The pre-processing steps of the training pipeline could be improved on as well, as some kind of pre-processing seems to be necessary to clean the data and make it more balanced, but the chosen methods lead to a reduction of accuracy.

Another idea which can be tested in the future, but only in combination with a larger dataset, is the inclusion of BIO-tagging of the dataset. As of now, it is possible for a code to be assigned to multiple tokens, but the code can only be represented as the same category being assigned to multiple tokens individually. The introduction of a start- and end-tag to a code increases the amount of tags significantly, but is a more correct representation of codes spanning multiple tokens, and could further increase accuracy. On the other hand, if there are only small amounts of data, the classification of TORE-categories could be reduced to the classification of TORE-levels, as this would reduce the number of categories from 12 to three. The classification of datasets into the levels could still be helpful for requirements engineers, and the accuracy may increase for the classification algorithm.

7 Bibliography

[1] M. Anders, M. Obaidi, B. Paech, & K. Schneider, "A study on the mental models of users concerning existing software," in *Requirements Engineering: Foundation for Software Quality*, V. Gervasi & A. Vogelsang, Eds., Springer International Publishing, 2022, pp. 235–250. DOI: 10.1007/978-3-030-98464-9_18.

[2] D. M. Blei, A. Y. Ng, &. M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003. DOI: 10.1162/jmlr.2003.3.4-5.993.

[3] R. L. Brennan & D. J. Prediger, "Coefficient kappa: Some uses, misuses, and alternatives," *Educational and Psychological Measurement*, vol. 41, no. 3, pp. 687–699, 1981. DOI: 10.1177/001316448104100307.

[4] P. Devine, Y. S. Koh, & K. Blincoe, "Evaluating unsupervised text embeddings on software user feedback," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, 2021, pp. 87–95. DOI: 10.1109/REW53955.2021.00020.

[5] V. Efstathiou, C. Chatzilenas, & D. Spinellis, "Word embeddings for the software engineering domain," in *Proceedings of the 15th International Conference on Mining Software Repositories*, Association for Computing Machinery, 2018, pp. 38–41. DOI: 10.1145/3196398.3196448.

[6] J. L. Fleiss & J. Cohen, "The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability," *Educational and Psychological Measurement*, vol. 33, no. 3, pp. 613–619, 1973. DOI: 10.1177/001316447303300309.

[7] N. Gisev, J. S. Bell, & T. F. Chen, "Interrater agreement and interrater reliability: Key concepts, approaches, and applications," *Research in Social and Administrative Pharmacy*, vol. 9, no. 3, pp. 330–338, 2013. DOI: https://doi.org/10.1016/j.sapharm.2012.04.004.

[8] J. Grisham, S. Samtani, M. Patton, & H. Chen, "Identifying mobile malware and key threat actors in online hacker forums for proactive cyber threat intelligence," in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2017, pp. 13–18. DOI: 10.1109/ISI.2017.8004867.

[9] C. Guo, W. Wang, Y. Wu, N. Dong, Q. Ye, J. Xu, & S. Zhang, "Systematic comprehension for developer reply in mobile system forum," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 242–252. DOI: 10.1109/SANER.2019.8668016.

[10] H. U. Iftikhar, A. U. Rehman, O. A. Kalugina, Q. Umer, & H. A. Khan, "Deep learning-based correct answer prediction for developer forums," *IEEE Access*, vol. 9, pp. 128 166–128 177, 2021. DOI: 10.1109/ACCESS.2021.3108416.

[11] ISO/IEC 25010, *ISO/IEC 25010:2011, systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models*, 2011. DOI: 10.3403/30215101.

[12] J. A. Khan, Y. Xie, L. Liu, & L. Wen, "Analysis of requirements-related arguments in user forums," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*, 2019, pp. 63–74. DOI: 10.1109/RE.2019.00018.

[13] M. Li, L. Shi, Y. Yang, & Q. Wang, "A deep multitask learning approach for requirements discovery and annotation from open forum," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, Association for Computing Machinery, 2020, pp. 336–348. DOI: 10.1145/3324884.3416627.

[14] N. Li, L. Zheng, Y. Wang, & B. Wang, "Feature-specific named entity recognition in software development social content," in *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, 2019, pp. 175–182. DOI: 10.1109/SmartIoT.2019.00035.

[15] H. Liu, Q. Li, R. Yao, & D. D. Zeng, "Analyzing topics of juul discussions on social media using a semantics-assisted nmf model," in *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2019, pp. 212–214. DOI: 10.1109/ISI.2019.8823541.

[16] R. R. Mekala, A. Irfan, E. C. Groen, A. Porter, & M. Lindvall, "Classifying user requirements from online feedback in small dataset environments using deep learning," in *2021 IEEE 29th International Requirements Engineering Conference (RE)*, 2021, pp. 139–149. DOI: 10.1109/RE51729.2021.00020.

[17] B. Paech & K. Kohler, "Task-driven requirements in object-oriented development," in *Perspectives on Software Requirements*, J. C. S. do Prado Leite & J. H. Doorn, Eds. Springer US, 2004, pp. 45–67. DOI: 10.1007/978-1-4615-0465-8_3.

[18] S. Samtani, H. Zhu, & H. Chen, "Proactively identifying emerging hacker threats from the dark web: A diachronic graph embedding framework (d-gef)," *ACM Trans. Priv. Secur.*, vol. 23, 2020. DOI: 10.1145/3409289.

[19] M. Stade, M. Oriol, O. Cabrera, F. Fotrousi, R. Schaniel, N. Seyff, & O. Schmidt, "Providing a user forum is not enough: First experiences of a software company with crowdre," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, 2017, pp. 164–169. DOI: 10.1109/REW.2017.21.

[20] C. Stanik, M. Haering, & W. Maalej, "Classifying multilingual user feedback using traditional machine learning and deep learning," in *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, 2019, pp. 220–226. DOI: 10.1109/REW.2019.00046.

[21] J. Tizard, "Requirement mining in software product forums," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*, 2019, pp. 428–433. DOI: 10.1109/RE.2019.00057.

[22] J. Tizard, H. Wang, L. Yohannes, & K. Blincoe, "Can a conversation paint a picture? mining requirements in software forums," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*, 2019, pp. 17–27. DOI: 10.1109/RE.2019.00014.

[23] A. Wadhwani, P. Jain, & S. Sahu, "Injurious comment detection and removal utilizing neural network," in *2021 International Conference on Innovative Practices in Technology and Management (ICIPTM)*, 2021, pp. 165–168. DOI: 10.1109/ICIPTM52218.2021.9388331.

[24] X.-Y. Wang, X. Xia, & D. Lo, "Tagcombine: Recommending tags to contents in software information sites," *Journal of Computer Science and Technology*, vol. 30, pp. 1017–1035, 2015. DOI: 10.1007/s11390-015-1578-2.

[25] R. Williams, S. Samtani, M. Patton, & H. Chen, "Incremental hacker forum exploit collection and classification for proactive cyber threat intelligence: An exploratory study," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2018, pp. 94–99. DOI: 10.1109/ISI.2018.8587336.

[26] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, & S. Li, "Predicting semantically linkable knowledge in developer online forums via convolutional neural network," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, Association for Computing Machinery, 2016, pp. 51–62. DOI: 10.1145/2970276.2970357.

[27] X.-L. Yang, D. Lo, X. Xia, Z. Wan, & J.-L. Sun, "What security questions do developers ask? a large-scale study of stack overflow posts," *Journal of Computer Science and Technology*, vol. 31, pp. 910–924, 2016. DOI: 10.1007/s11390-016-1672-0.

[28] M. Yazdaninia, D. Lo, & A. Sami, "Characterization and prediction of questions without accepted answers on stack overflow," in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, 2021, pp. 59–70. DOI: 10.1109/ICPC52881.2021.00015.

[29] W. Zheng & M. Li, "The best answer prediction by exploiting heterogeneous data on software development qa forum," *Neurocomput.*, vol. 269, 2017. DOI: 10.1016/j.neucom.2016.12.097.

Glossary

Accuracy The ratio of correctly predicted items to the total number of items. 21

Annotation A version of an annotated dataset that is created by an annotator[1]. 2

Backend All parts of a software that are not seen by users. 32, 51, 52

Bug An imperfection or deficiency in a work product that impairs its intended use. 63, 64, 80

Classifier An algorithm which categorizes datasets. 2

Code An identifier that is assigned to a text passage. 2

Container Small, isolated environments, in which instances of applications can run. 10, 12, 80

Continuous Delivery An automated process, with which software changes can automatically be tested and published to a repository. 12

Continuous Integration An automated process, with which changes from multiple developers can be merged and tested. 9, 12

Dataset A collection of documents that are related in some way. 2

Deep Learning A type of machine learning, which uses neural networks with input, output and hidden layers. 4

F1-score The weighted average of Precision and Recall. 21

[1]This pdf-File was used for many requirement-related definitions: `https://www.ireb.org/content/downloads/1-cpre-glossary-2-0/ireb_cpre_glossary_de_2.0.1.pdf`

Frontend All parts of a software that users can see, like the user interface. 10, 32, 51, 52, 57, 63

Functional Requirement A requirement concerning a result or behavior that shall be provided by a function of a system. 32, 34, 43, 65

Mock-up A medium-fidelity prototype that demonstrates characteristics of a user interface without implementing any real functionality. 32, 49, 57

Natural Language A language that people use for speaking and writing in everyday life. 2, 8, 33, 89, 102

Neural Network Networks consisting of artificial neurons, which can be used for deep learning algorithms. 13, 18

Non-functional Requirement A quality requirement or a constraint. 32, 43, 65

One Hot Encoding A representation of categorical data as numerical data, in which categories are converted to vectors which are as long as the number of categories, and contain only one *1*. 92, 94, 95

Precision The ratio of correctly predicted positive items to the total number of positive items. 21

Recall The ratio of correctly predicted positive items to the total number of correctly predicted positive and incorrectly predicted negative items. 21

Requirement A documented representation of a need, capability or property. 1

Requirements Engineer A person who – in collaboration with stakeholders – elicits, documents, validates, and manages requirements. 1, 106

Snowballing Using reference or citations of sources, in order to find new papers.. 15, 16, 18

Text Passage A cohesive unit of speech of variable length, which contains relevant information for an annotator. 2

Virtual Machine Independent environment, which runs instances of applications and has its own complete operating system. 12

Acronyms

ACM Association for Computing Machinery. 17, 19

AI Artificial Intelligence. 18

API Application Programming Interface. 10, 24, 54

Bi-LSTM Bidirectional Long Short-Term Memory. 4, 14, 21

CNN Convolutional Neural Network. 17, 18, 21

CSDN Chinese Software Developer Network. 21

GUI Graphical User Interface. 75, 76

IEEE Institute of Electrical and Electronics Engineers. 17, 19

LDA Latent Dirichlet Allocation. 11

LSTM Long Short-Term Memory. 14, 21, 92

NFR Non-Functional Requirement. 43, 63

NLP Natural Language Processing. 18, 21, 25, 29, 90, 102

POS Part of Speech. 37, 89

RNN Recurrent Neural Network. 14, 17

SeaNMF Semantics Assisted Non-negative Matrix Factorization. 11

TDD Test-Driven Development. 64, 104

TORE Task-Oriented Requirements Engineering. 1

UI User Interface. 1, 5, 32, 33, 47

VM Virtual Machine. 12

WS Workspace. 47

List of Figures

List of Tables