Institute of Software Engineering
Software Quality and Architecture

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor's Thesis

Automated Issue Creation using Voice Recognition and Natural Language Processing

Fabio Schmidberger

Course of Study: Softwaretechnik

Examiner: Prof. Dr-Ing. Steffen Becker

Supervisor: Sandro Speth, M.Sc
Dr. rer. nat. Uwe Breitenbücher

Commenced: June 11, 2020

Completed: December 11, 2020

Abstract

In modern software development issues are very important for inter-team communication and project management. Issues are used to clearly state the requirements of a change request and allow teams to plan and track their tasks. As part of an agile development process new issues are discussed during the sprint planning and reviews. However, the creation of well-structured issues with current tooling is still very time-consuming. Elements like the title and body have to be typed into the issue card and labels have to be manually selected. This makes it difficult for product owners to create digital issues during a meeting. For this reason, product owners often create handwritten notes during meetings and create issues in the issue management system afterwards. The current process results in a time and location distance between the need to create an issue and the digital documentation of this issue. This makes the process inefficient and error-prone. A product owner is effectively documenting each issue twice, once on the sheet of paper and then again in the issue management system. This thesis introduces the concept of a digital voice assistant for issue management. This system aims to automate the issue creation process and allows a product owner or developer to freely dictate an issue. Based on the spoken input a structured issue is automatically created. Elements like the assignee, labels, and priority are extracted from free text. A speech recognition system and natural language processing will be used. While modern voice assistants like Google Assistant and Amazon Alexa are increasingly common in consumer households, the underlying technology is rarely used in the enterprise context to help automate administrative tasks. The concept developed in this thesis acts as a blueprint for systems to fill out domain-specific forms, like issues or bug reports. A prototype of the system was implemented to showcase its capabilities. The system follows a four-step process. In the first step, the spoken input is transcribed using a speech recognition system. In the second step, the transcribed text is annotated with a natural language processing toolkit. Based on the annotations and transcribed text a structured issue card is filled out in the third step. The user has the option to edit and confirm the result. Finally, the resulting issue card is passed into an existing issue management system (like Github or Gropius) over an API call. To validate this solution approach an experiment was conducted. Future research possibilities and potential new use cases of the system design are presented at the end.

Kurzfassung

In der modernen Softwareentwicklung sind Issues ein wichtiges Werkzeug im Projektmanagement und unterstützen bei der Team Kommunikation. In Issues werden Anforderungen von Change Requests und Bugs dokumentiert. Zudem nutzen Teams Issues, um ihre Sprints zu planen und den Fortschritt zu beobachten. Im agilen Entwicklungsprozess werden neue Issues oft während dem Sprint Meeting und bei Reviews diskutiert. Das Erstellen von digitalen Issues in aktuellen Issue Management Systemen ist allerdings zeitaufwändig. Nutzer müssen den Titel und die Issue Beschreibung manuell eintippen. Auch Elemente wie Labels, Assignee (der Zuständige für das Issue) und die Priorität müssen einzeln aus Menüs und Listen ausgewählt werden. Der aufwändige Prozess macht es schwierig für Product Owner Issues schon während des Meetings digital im Issue Management System zu erfassen. Aus diesem Grund halten Product Owner die besprochenen Issues während dem Meeting oft in handschriftlichen Notizen fest. Diese Notizen werden dann nach dem Meeting digital im Issue Management System erfasst. Dieser Prozess führt zu zeitlicher und räumlicher Distanz zwischen dem Bedürfnis ein Issue zu erstellen und der finalen digitalen Dokumentation des Issues. Das macht den Prozess ineffizient und fehleranfällig. Ein Product Owner dokumentiert jedes Issue effektiv zweimal: zuerst als Notiz auf einem Papierzettel und im Anschluss nochmal digital im Issue Management System. Diese Arbeit präsentiert das Konzept eines Sprachassistenten für die Issue Erstellung. Das System hat das Ziel die Issue Erstellung zu automatisieren und soll Product Ownern es erlauben Issues auch während Meetings frei ins System einzusprechen. Basierend auf dem gesprochenen Text wird ein strukturiertes Issue automatisch generiert. Elemente wie der Labels, Assignee und Priorität werden aus dem frei gesprochenen Text extrahiert. Automatische Spracherkennung und natürliche Sprachverarbeitung werden dazu eingesetzt. Digitale Sprachassistenten wie Google Assistant und Amazon Alexa werden immer häufiger von Privatanwender eingesetzt. Trotzdem kommt die zugrundeliegende Technologie kaum im Enterprise Bereich zum Einsatz um administrative Aufgaben zu automatisieren. Das in dieser Arbeit entwickelte Konzept stellt einen generalisierbaren Bauplan bereit für die Entwicklung von Systemen, die Formulare spezifischer Fachbereiche aus Spracheingabe automatisch ausfüllen können. Ein Prototyp des beschriebenen Konzeptes wurde implementiert, um die Funktionen präsentieren zu können. Das System folgt einem einfachen vier Schritte Konzept. Zuerst wird die gesprochene Eingabe des Nutzers durch ein Spracherkennungssystem transkribiert. Im zweiten Schritt wird die Transkription von einem natürlichen Sprachverarbeitungssystem analysiert und annotiert. Anhand der Transkription und der Annotationen wird im dritten Schritt eine strukturierte Issue Karte erstellt. Der Nutzer kann diese bei Bedarf bearbeiten und speichern. Zum Schluss wird das gespeicherte Issue über eine Schnittstelle in ein bestehendes Issue Management System übertragen, wie z.B. Github oder Gropius. Das Lösungsdesign wurde durch ein Experiment validiert. Am Ende der Arbeit wird ein Ausblick in weitere Forschungsmöglichkeiten gegeben und zusätzliche Anwendungsfälle der Lösung beschrieben.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Purpose and Scope . . . 3
1.3 Research Questions . . . 3
1.4 Thesis Structure . . . 4

2 Foundations **5**
2.1 Issue Management . . . 5
2.2 Automatic Speech Recognition (ASR) . . . 7
2.3 Natural Language Processing (NLP) . . . 11

3 Related Work **13**
3.1 Survey Procedure . . . 13
3.2 Semantic Recognition of Issues . . . 13
3.3 Domain Specific Speech Recognition . . . 14

4 Concept of the Issue Speech Assistant **17**
4.1 Requirements . . . 17
4.2 Overview of the Concept . . . 19
4.3 Architecture . . . 24
4.4 Speech Recognition Pipeline . . . 26
4.5 Natural Language Processing . . . 29
4.6 Issue Management System Integration . . . 31

5 Implementation **33**
5.1 Application Frontend . . . 33
5.2 Speech Recognition . . . 38
5.3 Natural Language Processing . . . 40
5.4 Application Deployment . . . 43

6 Evaluation **49**
6.1 Experiment Design . . . 49
6.2 Results . . . 50
6.3 Threats to Validity . . . 51

7 Conclusion and Future Work **53**
7.1 Results and Conclusion . . . 53
7.2 Future Research Opportunities . . . 54

Bibliography **57**

List of Figures

2.1 Architecture of ASR systems 8
2.2 CoreNLP Annotation Pipeline Architecture 12

4.1 Concept Design: UI Mockup 20
4.2 Concept Design: Main Steps 21
4.3 Service Architecture as a UML Component Diagram 25
4.4 Speech Pipeline 27

5.1 User Interface 34
5.2 User Interface: Speech Input 35
5.3 User Interface: Issue Card 36
5.4 Application Deployment Architecture 45

List of Tables

4.1 Stakeholder Interviews 17

6.1 Experiment Participants 49
6.2 Precision, Recall and F1 Score of the Issue Speech Assistant (ISA) system 50

List of Listings

5.1 Redux Dispatch Event . . . 37
5.2 Redux Dispatch Event . . . 37
5.3 Redux Reducer . . . 38
5.4 Web Speech API Consume Transcript . . . 40
5.5 Web Speech API Browser Compatibility Test . . . 40
5.6 CoreNLP Maven dependency . . . 41
5.7 CoreNLP Document Annotation . . . 41
5.8 CoreNLP Training Data Example . . . 42
5.9 Command to train CoreNLP Model . . . 43
5.10 Host SSH Configuration . . . 45
5.11 nginx.conf - Reverse Proxy Configuration . . . 46
5.12 docker-compose.yml deployment configuration . . . 47

Acronyms

ASR Automatic Speech Recognition. 7

ISA Issue Speech Assistant. xi

NER Named Entity Recognition. 11

NLP Natural Language Processing. 11

PWA Progressive Web Application. 36

1 Introduction

Digital voice assistants like Google Assistant or Alexa are getting increasingly common in our everyday life. In the last couple of years, they have seen an increase in user adoption. Today users can control their smart home appliances, music, and get news through these devices.

But for the general consumer, these digital voice assistants still have three major problems. Voice assistants have problems with feature discoverability, still do not have a high enough accuracy, and only offer limited functionality and value to the general user. [LB] presents this in further detail. It is often unclear to the user which features a voice assistant supports and what they can do. This is caused by the poor discoverability in voice interfaces since there are no buttons or a UI that can tell the user what the assistant can do. Voice assistants are still making a lot of mistakes, misunderstand individual commands far to often, and start the wrong actions despite a clear voice command. Even modern voice assistants provide only a very limited amount of functionality.

It seems that the three main issues of feature discoverability, poor recognition accuracy, and limited functionality are unique to consumer-focused systems. With an enterprise-focused voice assistant, feature discoverability is far less of an issue, because users get specific training on how to use the assistant. Employees also use the system for many hours a day, so they get used to the system's capabilities quickly. The speech recognition and natural language processing system can be optimized for a specific context and domain in which to operate, resulting in the possibility to increase accuracy. By helping to automate specific repetitive tasks that employees of an enterprise complete often, a lot of value to the user can be created despite a limited feature set.

Despite that, there are hardly any voice assistant systems that specifically target the enterprise and aim to support employees with their everyday tasks. Especially administrative tasks are often very time consuming and cumbersome to complete. [VKWM15] shows that using speech input could make it significantly faster to complete administrative tasks.

1.1 Motivation

In modern SCRUM based software development processes, product owners often have to enter dozens of issues after a sprint planning or review. During these SCRUM meetings, the team discusses new features, current bugs, and other tasks that have to be worked on during a sprint. The product owner has the responsibility to keep track of these elements and enter them into an issue management system. Creating new detailed issue entries in the issue management system, however, takes to long to be done immediately during the meeting.

Instead, the product owner often writes down short notes on a sheet of paper for these issues so they can be entered into an issue management system at a later time. This information transformation from a discussed issue to a handwritten note and then to a structured and digitized issue entry in the

issue management system is not ideal. It has the potential to cause information loss (when elements are forgotten to be added to the issue), fail to uncover misunderstandings (as the other stakeholders will only see the created issue hours after the meeting), and be very time-consuming. In the article "It's Not Just Standing Up: Patterns for Daily Standup Meetings"[1] on Martin Fowler's homepage, Yip highlights the importance of note-taking during these agile meetings.

It is very important for teams to carefully document their bugs, user stories, and tasks. Issue entries become an increasingly important tool for communication and project management, especially with an increase in remotely located development teams and teams across multiple timezones working on the same application together.

A system that would allow the team to efficiently create issue entries during such meetings could help to improve this process. The team members would no longer have to write extra notes for each issue they want to create after the meetings. All team members could immediately see the newly created issue and suggest changes if there was a misunderstanding or if elements are missing. The product owner would also no longer have to spend time after the end of the meeting to enter all of the required issues in the issue management system.

A digital voice assistant for issue creation has the potential to allow easy issue creation during SCRUM meetings. The product owner could speak freely to create the issue. The proposed system would use speech recognition and natural language processing to automatically recognize the elements of the issue and generate a structured issue. For example, a product owner could speak a user story freely into the microphone, and the system could automatically create the correct issue with title, nicely formatted markdown body, labels, and more.

The team in the SCRUM meeting can use this structured issue suggestion to discuss further details, add them to the issue and clear any misunderstandings. The product owner can then confirm the issue and it is automatically created in the issue management system.

To support this crucial part of the software engineering process the aim is to make it easier and faster to create well-structured issues. This system could streamline the process of issue creation and save time. Not only the issue creation process has inefficiencies. Also the issue management and team communication can be challenging when multiple teams and microservices are involved in one issue.

The cross-component issue management system *Gropius* proposed by Speth et. al. in [SBB20] aims to make the overall issue management for modern microservice oriented projects easier. This paper and his work in [Spe19] introduced the concept of cross-project issues to reduce the communication overhead for "issues affecting multiple projects or teams".

These cross-component issues still have to be manually created. In this thesis, a system will be proposed to automatically create (cross-component) issues based on spoken user input. The goal of the system is to reduce the effort and time it takes to create new and well-structured issues. This system also aims to be an example of an enterprise-focused speech recognition system, showing how a modern speech recognition system can be designed and optimized for limited domain-specific use cases in a professional context. For this reason, particular emphasis will be put on creating an extensible and customizable architecture.

[1] https://martinfowler.com/articles/itsNotJustStandingUp.html

A speech recognition service will be used to create text from spoken utterances. Natural language processing will be used to recognize the semantics of the text and create the correct issue accordingly.

For the natural language processing engine CoreNLP, as proposed by Manning et. al. in [MSB+14], will be used. CoreNLP is a well known and very mature natural language processing tool kit, that allows for easy training of domain-specific named entity recognition models.

This system will be able to support existing issue management platforms. Its goal is to augment the existing platforms and help to improve the issue management workflow. As part of this thesis the multi-project issue management system proposed by [SBB20] will be integrated.

Helping to automate the administrative tasks will free up time for important work and make the daily tasks more enjoyable.

1.2 Purpose and Scope

The purpose of this thesis is to design and develop a complete proof-of-concept system that allows automated creation of structured Cross-Component Issues from spoken text. This holistic approach requires research and development in the areas of Cross-Component Issue Management, microservice architectures, speech recognition, and natural language processing.

To be able to complete the research, design, and development work needed for such a system and still stay within the scope of a bachelor thesis, not every component can be researched and evaluated in full depth.

A completed proof-of-concept system is necessary to evaluate how useful such a system could be to end-users. Such an evaluation is critically important to determine if further research in this field is justifiable. This is the reason why modern software development approaches are used for this research and a full so-called Minimum-Viable-Product is developed as part of this thesis.

Rather than focusing solely on detailed research for one specific component, a proof-of-concept version of each component is developed (as described in Section 4.3). Therefore, particular care will be put into the future work section, and detailed approaches that could be taken to further the research in natural language processing and speech recognition for this field will be given.

1.3 Research Questions

This section describes the three fundamental research questions of this thesis. The first research question focuses on the speech recognition and natural language component of this thesis.

> **RQ 1**
> *How can Cross-Component Issues be recognized from the spoken text?*

For this research question, we will evaluate the structure of Cross-Component Issues and use CoreNLP to try to recognize the elements of these issues.

The second research question focuses on system development and architecture.

> **RQ 2**
> *How can a system that recognizes Cross-Component Issues from the spoken text be designed and architected?*

The goal of the second research question is to design and implement a proof-of-concept system that can recognize the elements of a Cross-Component Issue from spoken text and create an issue with these elements in a Cross-Component Issue Management System, e.g. Gropius.

The third research question focuses on the evaluation of the system.

> **RQ 3**
> *Can the system accurately create structured issues from the spoken text?*

The purpose of the third research question is to evaluate the system's performance and usability. The goal of this part is to evaluate if the proof-of-concept system can provide a good enough user experience to justify further research in the individual components of this thesis.

1.4 Thesis Structure

The thesis is structured as follows:

Chapter 2 - Foundations: This chapter presents issue management, speech recognition, and natural language processing as the foundations that are used in this thesis.

Chapter 3 - Related Work: related research to this thesis is presented in this chapter. Due to the novice concept developed in this thesis, a limited amount of related research is available at the time of writing.

Chapter 4 - Concept of the Issue Speech Assistant: In this chapter, the requirements of the target audience are analyzed. Based on the requirements an application concept is designed. Example use cases for this concept are presented and the architecture for the application is discussed. The purpose of each component of the microservice architecture is explained in detail.

Chapter 5 - Implementation: The implementation of the application concept presented in the previous chapter is discussed here. Tools and technology decisions are outlined. The chapter is concluded by showing the deployment architecture and automated deployment processes developed for this system.

Chapter 6 - Evaluation: This chapter evaluates the performance of the implemented application. An experiment with multiple participants was conducted to determine the recognition accuracy. Threats to the validity of the experiment results are discussed at the end of the chapter.

Chapter 7 - Conclusion and Future Work: The results of this thesis are summarized and opportunities for future research are presented.

2 Foundations

This chapter describes the foundations used in this work. First, Section 2.1 describes issues, issue management and the concept of cross-component issues. Afterwards, current speech recognition frameworks are compared in Section 2.2 as one of them will be used for the implementation in this thesis. Section 2.3 outlines the basics of natural language processing.

2.1 Issue Management

This section describes issue management in detail as it is a foundational part of this thesis. First the term *issue* is defined in Section 2.1.1. In Section 2.1.2 issue management systems and their uses are described. The lifecycle of an issue in an issue managament system is detailed in Section 2.1.3. Finally the concept of cross-component issues is discussed in Section 2.1.4.

2.1.1 Issues

In modern software development issues are used to document any kind of change request or task related to the project. These change requests can include bug reports, feature requests or refactorings. Developers and other stakeholders of a software product can create new issues and communicate their ideas with the team, as outlined by Sommerville

In tools like Github[1] and Gitlab[2] developers can create issues in a code repository.

Issues have a simular structure in both tools. An Issue is part of a project. Each issue has a title, body, assignees, and labels. In the following, these elements of issues are explained in further detail. The structure and datatypes of each element is important.

For the purpose of this evaluation, only Github and Gitlab were used. For more details on other issue management systems such as Atlassian Jira[3] or Readmine[4] see [Spe19].

1. **Project:** When creating a new issue it is part of a project. Issues are generally not intended to be moved between different projects. In systems like Github or Gitlab a project has a single repository.
2. **Title:** The title is a single-line text field intended for a short and descriptive text about the intention of the issue. To create an issue a title is required on both Github and Gitlab.

[1]https://github.com/
[2]https://gitlab.com/
[3]https://www.atlassian.com/de/software/jira
[4]https://www.redmine.org/

3. **Body:** The body allows the developer to add additional information and describe the issue in further detail. The body mostly contains free text written in markdown format.
4. **Assignees:** The assignees are a list of users that are responsible for the issue. Both Github and Gitlab support multiple assignees. In Jira, issues are designed to have a single assignee[5].
5. **Labels:** A list of labels that are used to categorize and filter issues.
6. **Milestone:** A milestone lets a team group multiple issues together and allows the team to track the combined progress.
7. **Weight:** Estimated complexity of the issue. Commonly measured in story points. (Isuse weight is not supported by Github).
8. **Due date:** The date on which the issue has to be closed at the latest.
9. **Comments:** After the issue was originally created other developers can comment on a issue to provide feedback.

2.1.2 Issue Management Systems

The purpose of an issue management system is to manage and maintain multiple issues. They allow team members to edit, update, and comment on issues. Isuse managament systems give users an overview of all issues and can notify users if particular issues are updated.

Issue management also often include progress tracking features that allow teams to analyze if they are reaching their goals. Gitlab for example includes burndown charts[6] that lets a team see if they are still on track to complete their milestone on time.

2.1.3 Issue Management Lifecycle

On the surface, issues have a really simple live cycle. In issue management systems that are part of tools like Github or Gitlab an issue can either be `open` or `closed`. An open issue represents a task that is not yet completed. A closed issue represents a task that has been completed and no longer has to be worked on. An issue can be `reopened` if a previously closed issue needs to be worked on again. This might be the case if a team thought that a bug had been fixed by a particular merge request and then the same bug resurfaced after the issue had already been closed.

An issue goes through the steps of ideation, creation, discussion, planning, implementation, review to closed. As most issue management systems only have the two states of open and closed for their issues, teams use labels to describe in which stage the issue currently is.

[5] Issues can only be assigned to multiple assignees using workarounds: https://confluence.atlassian.com/jira/how-do-i-assign-issues-to-multiple-users-207489749.html

[6] https://docs.gitlab.com/ee/user/project/milestones/burndown_and _burnup _charts.html

2.1.4 Cross-Component Issues

The adoption of microservice architectures lead to teams managing multiple microservices rather than a single monolith. In a microservice architecture, a single feature request can easily require changes in multiple services. This can make issue management particularly difficult. Tracking bugs and change requests across multiple components is not properly supported by current tooling. Existing issue management systems like the ones in Github, Gitlab or Jira do not allow issues to be part of multiple projects.

Some teams use a monorepo approach, where all components are maintained in a single git repository. However, the monorepo is not widely adopted [Bro19] and even strong proponents of monorepos like Google conclude in a case study [JJK+18] that the use of is monorepos is still a "comparison of tradeoffs". Organizations like Amazon and Netflix are firmly staying behind their polyrepo approach [Bro19]. This makes it clear that there is a need for cross-component issues that allow the use of polyrepo structures.

Speth identified in [Spe19] the problem that current issue management systems as described in Section 2.1.2 are not sufficient to manage issues for applications based on the increasingly common microservice architecture. He proposed a new issue management system optimized for these component-based architectures with support for cross-component issues. Speth et. al. build in [SBB20] on this work and present Gropius, a cross-component issue management tool.

In [Spe19] the term "multi-project coding issues" was used. However, in more recent work from S. Speth et. al. "multi-project coding issues" have been renamed to "Cross-Component Issues".

The cross-component issue management system Gropius allows for visualization of the component-based architecture and displays the components affected by an issue. This visualization helps teams to manage and track the cross-component issues. Gropius uses integration adapters to sync issues between multiple issue management systems. This has the significant advantage that teams can continue to use their existing tooling.

2.2 Automatic Speech Recognition (ASR)

With Automatic Speech Recognition (ASR) spoken input from a user can be transcribed. Speech recognition takes the audio input and generates a written transcription. This is also called speech-to-text or in short SST. In speech synthesis, written text is used to create spoken words. Creating spoken words from written text is called text-to-speech or TTS. In this thesis, the focus will be on speech recognition. Speech synthesis will not be used.

As described by Yu et al. in [YD16], a typical ASR system consists of four main components: signal processing and feature extraction, acoustic model, language model, and hypothesis search. These main components of a ASR system are illustrated by Figure 2.1.

The signal processing and feature extraction component processes the audio input, extracts the audio features as vectors and passes them on to the acoustic model. An acoustic model is trained with audio data that contains recorded speech and the matching text transcriptions to that audio data. This results in a statistical model of the sounds that are part of each word.

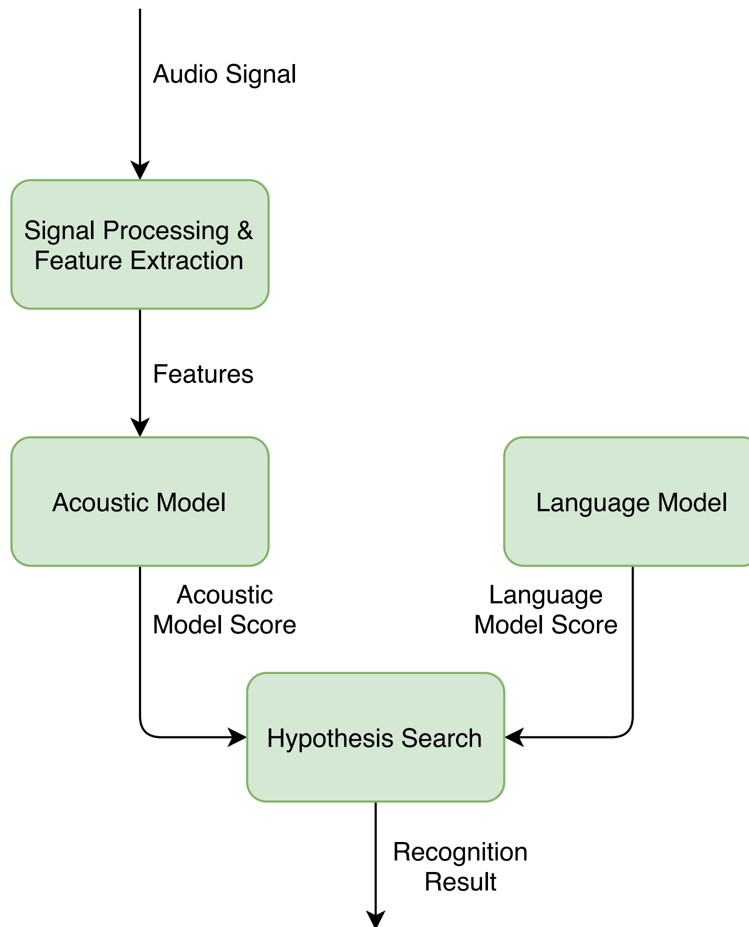


Figure 2.1: Architecture of ASR Systems (based on illustaration from [YD16])

The language model (LM) estimates the probability of a given word sequence. Language models is trained on text corpora and learns the correlation between words in the training data. The accuracy of the language model can be improved if it is only intended for use in a specific domain. By training the language model on text data that is specific to the intended domain it can be less accurate in general-purpose tasks but more accurate on for its intended domain. The concept of a speech recognition system for issues presented in this thesis would be a great use case for a domain-specific language model.

The resulting scores from the acoustic model and language model are combined in the hypothesis search component. The hypothesis component outputs the word sequence with the highest probability. This is the recognized result.

2.2.1 Speech Recognition Toolkits

There are multiple different tools you can use to add speech recognition capability to your application. In the following the popular open-source speech recognition toolkits DeepSpeech, wav2letter++ and Kaldi are presented. A web application can also use the Web Speech API to add speech recog-

nition capability for the user. This is an alternative to implementing a separate speech recognition microservice. The Web Speech API was used in the final system of this thesis, as DeepSpeech had accuracy issues.

DeepSpeech (recently renamed to Mozilla Voice SST)

DeepSpeech[7] is an open-source speech-to-text engine. The implementation is based on Baidu's Deep Speech research paper [HCC+14]. DeepSpeech is implemented with Google's TensorFlow machine learning framework.

Recently the DeepSpeech research project was renamed by the Mozilla foundation to Mozilla Voice STT[8] (SST stands for *speech to text*). However, Mozilla is not very consistent with its branding and still frequently uses the name DeepSpeech.

Both the documentation[9] and Github repository[10] still use the name DeepSpeech, rather than the new name Mozilla Voice STT. At the time of writing DeepSpeech, is far more commonly used name for this speech recognition toolkit. In order to keep consistency, the name DeepSpeech will be used exclusively throughout this thesis.

wav2letter++

wav2letter++[11] is an open-source speech recognition toolkit from Facebook AI Research. It is written in C++. Wav2Letter++ is the youngest of the speech recognition tools discussed here and was open-sourced in December of 2018. First results of wav2letter++ seem very promising. Zamia reports a word error rate of less than 4 percent[12].

Kaldi

Kaldi[13] is an open-source speech recognition toolkit developed by its community. It is written in C++. Kaldi is licensed under Apache v2.0. The Kaldi Speech Recognition Toolkit is described in detail in [PGB+11].

[7] https://voice.mozilla.org/stt.html
[8] https://voice.mozilla.org/stt.html
[9] https://deepspeech.readthedocs.io/en/v0.9.1
[10] https://github.com/mozilla/DeepSpeech
[11] https://github.com/facebookresearch/wav2letter
[12] https://goofy.zamia.org/asr/
[13] https://kaldi-asr.org/

There are pre-trained models for different languages and trained on different corpora available. Particularly good models for Kaldi are available from `Zamia`[14]. These models are free and open-sourced on Github[15]. Zamia provides models in German and English. For the English models, Zamia reports a Word-Error-Rate between 5.8% and 10.6%[16]. For the German models, Zamia reports a Word-Error-Rate between 8.4% and 11.5%[17].

In contrast to DeepSpeech described in Section 2.2.1 and Wav2Letter++ described in Section 2.2.1, Kaldi does not implement an end-to-end speech recognition pipeline using a deep neural network. Kaldi uses more conventional GMM (Gaussian Mixture Models) and SGMM (Subspace Gaussian Mixture Models) acoustic models[18].

Web Speech API

The Web Speech API is a JavaScript API included in the browser which allows developers to easily enable speech recognition and text-to-speech functionality in their web applications. It is important to keep in mind that the Web Speech API is not a speech recognition tool kit. Instead, it is a uniform API you can use to take advantage of the speech recognition system that the browser manufacturer implemented.

The speech recognition functionality is provided by the `SpeechRecognition` interface and the `SpeechSynthesis` interface can be used for text-to-speech. It is important to keep in mind that the Web Speech API is currently not a W3C Standard nor is it on the W3C Standards Track. The Web Speech API specification[19] was published by the Web Platform Incubator Community Group.

The Web Speech API abstracts the entire speech recognition system away and allows developers to call browser integrated functionality rather than implementing the speech recognition service themselves. As the Web Speech API is only in draft status there is currently limited browser support. Google Chrome already added support for the Web Speech API in 2013[20].

A major drawback is that the Web Speech API is currently only supported by Google Chrome browsers. As reported by Mozilla[21] the Web Speech `SpeechRecognition` interface is only implemented by Google Browsers including Firefox, Opera and Safari do not support the Web Speech API at the time of writing.

Another drawback when using the Web Speech API is that users do not have control over where and how the audio data is being processed. The current Google Chrome Browser implementation of the Web Speech API for example uses a server-based speech recognition engine and utilizes Google Cloud services.

[14] https://zamia.org/asr/
[15] https://github.com/gooofy/zamia-speech
[16] https://zamia.org/asr/
[17] https://zamia.org/asr/
[18] http://www.kaldi-asr.org/doc/model.html
[19] https://wicg.github.io/speech-api/
[20] https://developers.google.com/web/updates/2013/01/Voice-Driven-Web-Apps-Introduction-to-the-Web-Speech-API
[21] https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API

This means that all the audio recorded by your application will be sent to a remote web service for speech recognition. Depending on the intended use cases this can be a big problem from a data privacy perspective. The use of a remote speech recognition service also means that the functionality will not be available to your application when running in offline mode. Especially with the increase in popularity of Progressive Web Applications, offline capability can be quite important.

To summarize, the Web Speech API offers a fast and easy way to add speech recognition and speech synthesis functionality to your web application. The implementation on Google Chrome browsers has very high accuracy.

2.3 Natural Language Processing (NLP)

Natural language processing (NLP) was defined by Chowdhary in [Cho20] as "the study of computer systems for understanding and generating natural language". For this thesis, the focus will be on "understanding" natural language.

Popular Natural Language Processing (NLP) toolkits are SpaCy[22] and CoreNLP[23]. CoreNLP is an open-source NLP toolkit developed by Stanford and written in Java. SpaCy is written in python and also open-source. With SpaCy's python background it is easier to integrate it into existing deep-learning pipelines.

NLP tools have a pipeline with multiple steps like tokenization, parts-of-speech, and named entity recognition to achieve their annotation results. They take raw text as input, execute their annotation steps, and return annotated text. Figure 2.2 visualizes the annotation pipeline for CoreNLP. The individual steps of the annotation pipeline are explained in detail.

In the Tokenization step, a tokenizer splits the input text into a sequence of tokens [MSB+14]. A token can be a word, punctuation, or number. A sentence like "This Apple won't sell for 300€." will be split into the following tokens (CoreNLP was used for this example): "This", "Apple", "wo", "n't", "sell", "for", "300", "€", ".". The Sentence splitting step takes the sequence of tokens and groups them into sentences. All tokens that are in one sentence will be grouped together.

In part-of-speech (POS) tagging every token gets labeled as a noun, pronoun, verb, adjective, adverb, preposition, conjunction, and interjection. In lemmatization, each token is assigned its lemma. For example the words "were", "is", "are", "been" all have the same lemma "be".

2.3.1 Named Entity Recognition (NER)

Named Entity Recognition (NER) is one of the fundamental components in natural language processing. With NER elements in a sentence can be classified into specific groups. A named entity represents a normal object that is referenced in the text. This could be something like a name, city, number, component, or label. NER is able to detect these named entities in a written text. This helps to extract the information from the text.

[22]https://spacy.io/
[23]https://stanfordnlp.github.io/CoreNLP/

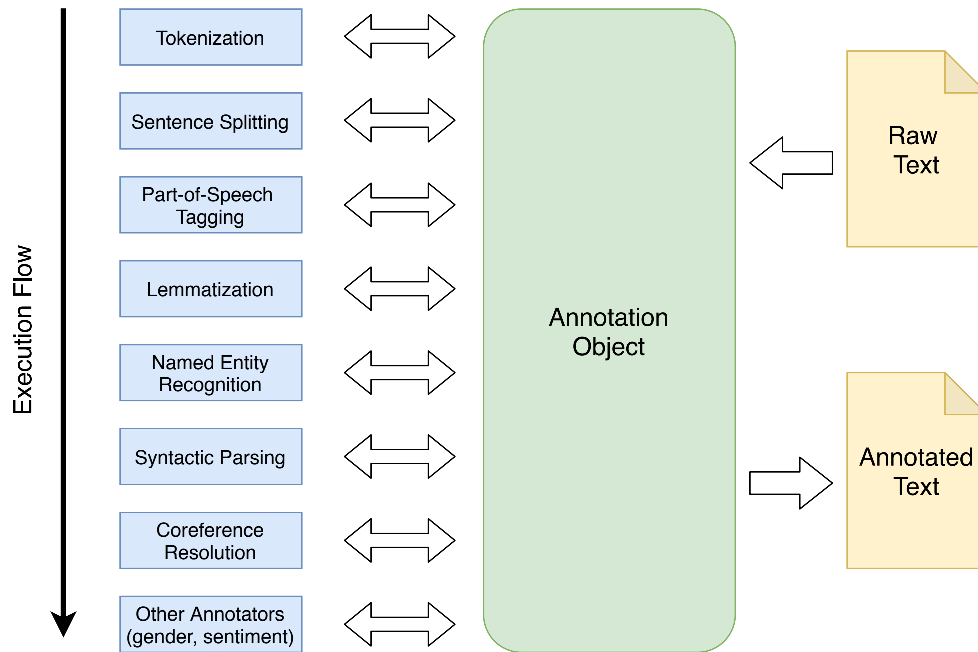


Figure 2.2: CoreNLP Annotation Pipeline Architecture (based on illustaration from [MSB+14])

NER can be split into two separate problems. In the first steps, all the tokens that belong to one name have to be detected. This is called `name detection`. A name like "New York" for example consists of two tokens.

In the second step, the detected names have to be classified into predefined entity groups. To classify the detected names into the correct entity groups, a model can be used. The model is trained based on a data set of sentences with labeled entities.

3 Related Work

In this chapter, related work to the topics of semantic analysis of issues and domain-specific speech recognition is reviewed. First, the survey procedure that was used is presented in Section 3.1. Section 3.2 describes research that has been done in automatically extracting information from unstructured issues. Related work on domain-specific speech recognition systems is discussed in Section 3.3.

3.1 Survey Procedure

The search engine Google Scholar[1] was used primarily to find relevant research papers.

To find work related to this thesis, the following survey questions were defined:

> **SQ 1**
> *How can semantic information be extracted from issues?*

> **SQ 2**
> *How can speech recognition be optimized for specific domains?*

During the research efforts the following key-words were used: semantic recognition of issue structure, bug analysis, semantic analysis bug reports, semantic analysis issues, speech recognition bug reports, domain-specific speech recognition, speech recognition optimized for domains, speech recognition optimized for use cases .

Due to the novice nature of the concept presented in this thesis not a lot of related research has been conducted at the time of writing. In this thesis the existing approach of speech recognition and natural language understanding has been applied to issue management.

3.2 Semantic Recognition of Issues

Existing research in this area primarily focuses on detecting and classifying bug reports. Zhou et. al. developed a system in [ZLSG18] to recognize “bug-specific named entities”. In their work they designed and implemented their own bug-specific NER system and named it `BNER`. They contributed

[1] https://scholar.google,com

a comprehensive classification of different categories for bug specific entities. To get training data for their NER system they gathered solved bug reports from the Mozilla and Eclipse projects on Bugzilla. A total sample of 1800 bugs were randomly selected and labeled by a team of 8.

Chen et. al. builds on top of the work from Zhou et. al. in [CLZZ19] and propose a neural network to extract bug entities and their relationships. In their work Chen et. al. relied in [CLZZ19] on the Stanford CoreNLP toolchain for the natural language processing. The same tool will also be used in this thesis.

Ye et. al. also designed a NER system specific for software engineering in [YXF+16]. Instead of labeling bug reports as Zhou et. al. in [ZLSG18] Ye et. al. focus on analyzing posts in social communities such as Stack Overflow[2]. Ye et. al. highlights that "one must first understand the unique characteristics of domain-specific texts". The system developed by Ye et. al. was called S-NER. In their work they labeled over 1500 Stack Overflow posts for the supervised learning of the NER system. Using an iterative approach Ye et. al. improved the pipeline of their S-NER annotator.

Related work that focuses on the semantic recognition of issues from spoken input has not been found during the literature review procedure. These designs extracted information from issues that have already been created in the issue management system. Rather than developing an own specialized NER system like Ye et. al. or Zhou et. al. the NER engine in the open-source natural language processing toolkit CoreNLP will be used in this thesis. In their work Ye et. al. and Zhou et. al. showed that text with terminology that is specific to the software development domain can be accurately classified using NER techniques. For the training of their classifiers, they both used over 1500 labeled data entries, which were annotated by hired annotators.

3.3 Domain Specific Speech Recognition

Current state-of-the-art speech recognition systems like DeepSpeech[3], Kaldi[4], Amazon Transcripe[5], or Google Speech API[6] are general-purpose speech recognition system. They are trained on very large datasets of general-purpose speech corpus, like the Common Voice Corpus[7]. Ardila et. al. presents the data collection procedure and text corpus of Common Voice in [ABD+19]. The Common Voice website allows for crowd-sourced collection and verification of audio data.

Models public state-of-the-art speech recognition systems are trained on a collection of this non-domain specific audio data. The acoustic and language model used in these systems is optimized for general-purpose recognition and not for a specific domain.

This makes these general-purpose speech recognition systems a great allrounder and allow for usage in many different use cases. However, there are many specific domains that have a specific vocabulary and specific sentence structures that are frequently used. A well-known example of

[2] https://stackoverflow.com/
[3] https://github.com/mozilla/DeepSpeech
[4] https://kaldi-asr.org/
[5] https://aws.amazon.com/de/transcribe/
[6] https://cloud.google.com/speech-to-text
[7] https://commonvoice.mozilla.org/

specific domains with custom vocabulary is the medical field. General-purpose speech recognition systems have poor accuracy for medical terms and are only usable in the medical field if properly optimized. The domain of issue creation is also very specific. Issue creation requires the use a lot of technical terms primarily used by developers like API, microservice, backend, bug, stack trace, and many more. Just like in the medical field existing general-purpose speech recognition systems have limited accuracy which harms the user experience. A speech recognition system optimized for the specific domain would be needed for optimal results.

There are two different approaches to optimizing a speech recognition system for a specific domain. The first approach would be to train a custom acoustic and language model for a specific domain. Training custom models is only supported by speech recognition frameworks like DeepSpeech, Cloud services for speech recognition like Amazon Transcribe and Google Speech API do not support custom models that are provided by a developer. This approach requires new optimizations of the acoustic and language model for each new domain and language that has to be supported, which can be very expensive.

The second approach would be to take use a readily available general-purpose speech recognition system and develop a post-processing system that can adapt and repair erroneous transcriptions from the speech recognition. This design was presented by Anantaram et. al. in [AK17]. The speech recognition gets a spoken input S and generates a transcription output of T'. The correct transcription would be T. As part of his work Anantaram et. al. developed "two mechanisms for adaption or repair of the ASR output, namely $T' \longrightarrow T$."

The research survey showed that the trend and research focus seems to go towards general-purpose speech recognition systems and optimizing their performance on very large datasets. Limited research is done in the field of domain-specific speech recognition systems. The high cost of collecting the required domain-specific labeled audio data and optimizing the speech recognition for the domain might be the reason for it. Models can not be reused across distinct domains or languages without which further increases the cost.

4 Concept of the Issue Speech Assistant

The following chapter will give an overview of the application concept that was designed for this thesis. First, the necessary requirements of the application are evaluated in Section 4.1. Based on these requirements the application design was created. This is presented in Section 4.2. Possible example uses of the concept are outlined in Section 4.2.3. The architecture of the application is discussed in Section 4.3. Furthermore the speech pipeline, natural language processing system and integration design are described in Section 4.4, Section 4.5, and Section 4.6.

4.1 Requirements

In this section, the requirements of the application are outlined. First, the requirements engineering process used in this thesis is described. After this the persona of an intended typical user is shown. To conclude this section the list of gathered requirements is given.

4.1.1 Requirements Engineering Process

As a first step in the requirements engineering process possible stakeholders where identified. In the issue management process the roles of product owner, software developer, software architects, and (non-technical) domain experts are stakeholders. Individuals that work in these roles were contacted and interviewed. The problem statement was presented to the interviewees. The interviews were conducted over the phone, video calls, in-person, and instant messengers like Telegram.

For each of the stakeholders, multiple individuals were interviewed as shown in Table 4.1.

Based on these interviews the product owner was identified as the ideal target user for the application developed in the thesis. Of the interviewees, the individuals working as product owners had the most pain points in the current issue management process. It also got apparent that individuals who are working in the role of product owner spent the most time on issue management and created the most issues. While the product owners that were interviewed created on average 43 issues per week,

Stakeholder	Number Interviewees
product owner	3
software developer	2
software architect	2
(non-technical) domain expert	1

Table 4.1: Stakeholder Interviews

developers and software architects both created less than 15 issues every week. The (non-technical) domain expert that was interviewed only occasionally (less than 5 per week) created an issues in the team's issue management system.

It should be noted that due to the limited sample size of these interviews these results do not carry a lot of scientific weight. Additionally, the majority of the interviewees worked in small software development organizations with a strong focus on agile processes. The individual development process of these organizations might affect their use of their issue management systems and the interview results. However, these results are sufficient to continue the requirements engineering process for the prototype aimed to be developed as part of this thesis. Further research would be required before a real product could be developed.

Based on this evaluation, the application developed in this thesis will target product owners and focus on the requirements of product owners. In the next section, a persona for a typical product owner is developed. This helps to contextualize the concrete requirements that were later collected.

4.1.2 Personas

Based on the interview a persona for the intended user was developed. Personas are fictional characters, that are created to represent specific user types.

In the following the persona of Peter, the product owner is introduced. He represents the type of users the further application design will focus on.

Peter - Product Owner

Peter is 35 and the product owner in a software development team of seven. He and his team are building a SaaS software product. They are using SCRUM as their development process with two-week sprints. As product owner Peter is responsible for maintaining the team's backlog.

In their project management process, they aim to create issues that are small enough so that they can be completed within one day. Most issues can be completed in a couple hours. In an average sprint of two weeks the team completes roughly 100 issues. About 80 of these issues are created by Peter. The idea or need for most of these issues comes up during the sprint planning and sprint review. Currently, Peter takes notes on sheets of paper during the meeting and creates new issues in the issue management system after the meeting.

Each issue describes a change request with limited scope. Often Peter only adds a title, labels, priority, and assignee to the issues. A detailed description of the issue in the body is rarely required. As they are a small team they often discuss details during spontaneous meetings.

Peter is unhappy with the current process, where he writes notes during the meeting and has to type them into the issue management system after the meeting. Sometimes he forgets to create issues he and his team discussed during a meeting.

4.1.3 Gathered Requirements

As a result of the interviews and requirements engineering process, the following requirements were gathered. These requirements are written as user stories with the persona of Peter as the intended user.

These userstories have the following structure: "As a *type of user*, I want *achive some goal* so that *reason*."

- As a product owner, I want to create issues efficiently during a meeting, so that I don't have to write them on paper during the meeting and digitize them later.
- As a product owner, I want to show the content of the issue to the entire team during the meeting, so that we can check if there are any misunderstandings in the team.
- As a product owner, I want to efficiently create multiple issues in succession, so that I can easily document bugs in the review process.
- As a product owner, I want to see what elements in the spoken input have been recognized so that I can quickly identity if anything is missing.
- As a product owner, I want to reset the current input, so that I can remove text with errors in it.
- As a product owner, I want to be able to connect the application with the issue management system that I currently use.
- As a product owner, I want to be able to use the application on any laptop with no installation required, so that I can use for example the laptop of a college in the meeting to create the issues.
- As a product owner, I want to be able to edit issues, so that I can fix any mistakes that I previously made.

4.2 Overview of the Concept

Based on the requirements outlined in Section 4.1.3, an application concept was designed.

The application should be a web application that allows a user to dictate the issue they want to create. The web application shows the spoken input text on the left side of the screen. The speech recognition results are displayed immediately while the user is still speaking. Once the user stops the recording the recognized text is analyzed by a natural language processing system. A separate microservice will be responsible for this analysis.

The results of the natural language processing are visualized by highlighting the recognized entities. Based on these results an issue card on the right side of the screen is filled out. The issue title and issue body should be filled out based on the spoken user input. Elements like the assignees, labels, and components are recognized as well and shown in the created issue card.

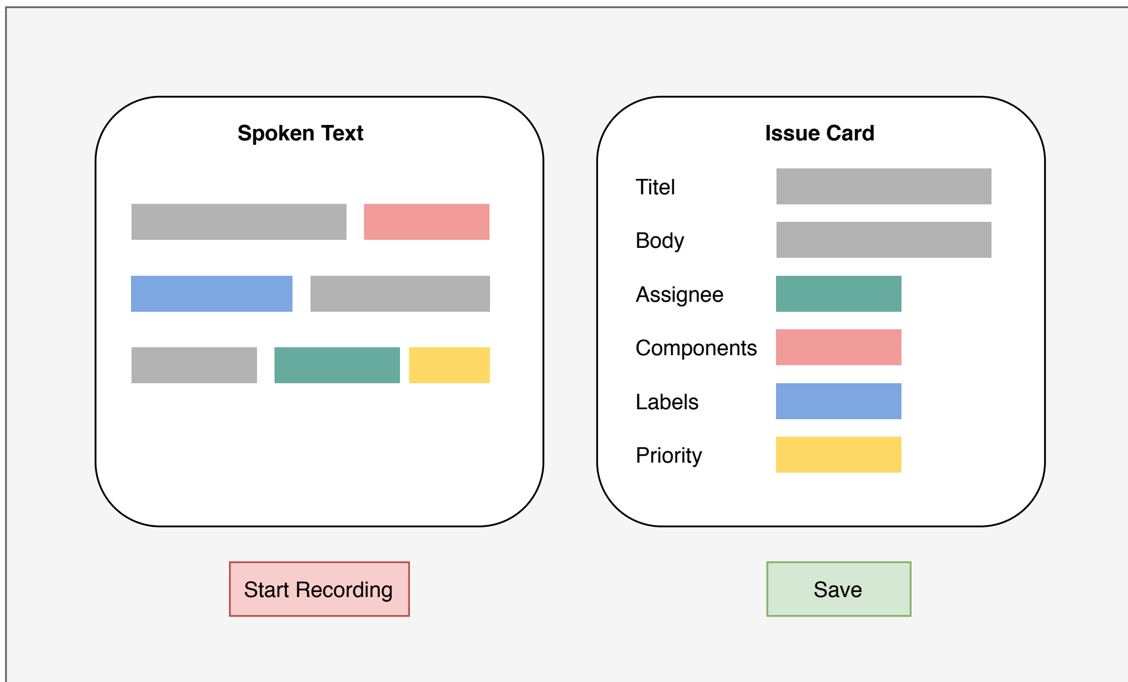


Figure 4.1: Concept Design: UI Mockup

The automatically generated issue is reviewed by the user. If anything is incomplete or was not recognized correctly the user manually corrects the issue. Once the issue is completed the user can press a save button. When the user presses the save button the issue with all of the filled out data elements should be automatically created in the connected issue management system.

Figure 4.1 shows a basic mockup of the user interface, that was designed for this concept.

The user should have the ability to connect different issue management systems. Github and the Cross-Component Issue Management System *Gropius* developed by Speth et. al [SBB20] should have integrations. The user can configure which issue management system is connected in the settings menu of the application. The configuration of a user will be saved.

From the users perspective the following steps will be completed:

1. The user opens the application in a web browser.
2. The user presses the "start recording" button and speaks freely into a microphone.
3. The spoken text is converted into written text using speech recognition.
4. The written text is analyzed using natural language processing. The content of the spoken text is understood.
5. Using the results of the natural language processing a structured issue is created. Elements like title, labels, a text body and assignee are filled out automatically.
6. The user can confirm or edit the proposed issue.
7. The issue is automatically created in existing issue management systems.

This user journey can be reduced to four crucial steps. In the first step, speech recognition is used to create a transcript of the spoken input. In the second step, the transcription is analyzed by a natural language processing engine. The results from the speech recognition and natural language processing in step one and two are used to create a structured issue in step three. Once the user confirms the structured issue it is synchronized with an existing issue management system in step four. These steps are visualized in Figure 4.2.

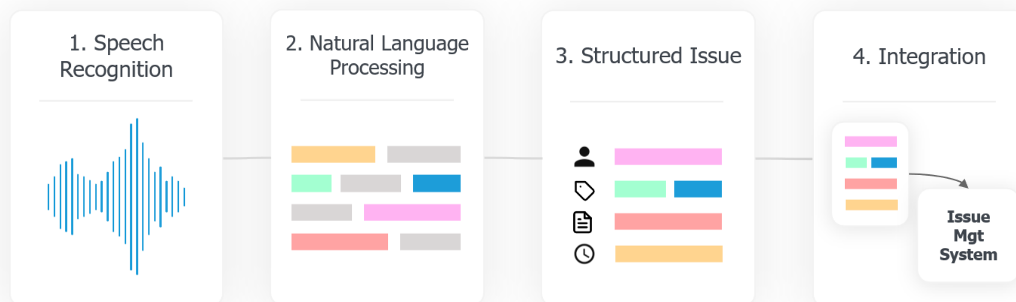


Figure 4.2: Concept Design: Main Steps

4.2.1 Application Name

The application needs a name so it can be referenced throughout the rest of this thesis and in future work. Requirements for the name of the application include:

- should reflect the purpose of the application: based on the application name a user should be able to recognize that the application is a speech recognition for issues
- short
- easy to remember
- easy to write
- can be spoken
- low chance of conflict with other existing applications or well-known concepts

Based on these criteria the following names were part of the final selection:

- ISR: Issue Speech Recognition
- AISR: Automated Issue Speech Recognition

- SIRA: Speech Issue Recognition Automated
- ISA: Issue Speech Assistant
- TSR: Task Speech Recognition

From this selection, the name ISA was chosen. It is short and memorable. Isa is also a first name, which helps to humanize the software system developed here. Other digital assistants like Alexa and Siri have chosen a similar naming strategy and picked names that are also similar to a first name. In contrast to some of the other options like AISR, ISA can easily be spoken.

In the following sections, the application presented in this thesis will be referenced as ISA.

4.2.2 Design Limitations

The concept of ISA has limitations that have to be considered. In this section, the usability definition from Nielsen introduced in [Nie94] will be used to evaluate the concept. Nielsen defines usability as a combination of the so-called "usability attributes": Learnability, Efficiency, Memorability, Errors, and Satisfaction. These terms are defined in [Nie94]:

- **Learnability:** The system should be easy to learn so that the user can rapidly start getting some work done with the system.
- **Efficiency:** The system should be efficient to use, so that once the user has learned the system, a high level of productivity is possible.
- **Memorability:** The system should be easy to remember, so that the casual user is able to return to the system after some period of not having used it, without having to learn everything all over again.
- **Errors:** The system should have a low error rate, so that the users make few errors during the use of the system, and so that if they make errors they can easily recover from them. Further, catastrophic errors must not occur.
- **Satisfaction:** The system should be pleasant to use, so that users are subjectively satisfied when using it; they like it.

Learnability is one of the core weaknesses of voice-enabled systems. Users have to learn which terms and phrases can be used. It is not immediately apparent what the supported features are. In typical applications with a graphical user interface (GUI), the user can simply look at the available buttons and determine the available functions. With a voice user interface these functions are not immediately apparent. Feature discovery is a real challenge for users. Users have to effectively experiment what works. These same aspects also cause reduced memorability. A guided tutorial could help with the learnability of the system.

These usability limitations are part of the inherent design of voice assistant systems and are present in most digital voice assistants on the market like Google Assistant[1], Amazon Alexa[2] and Siri[3]. These systems target general consumers so the learnability and intuitiveness of the system is really important. As the ISA system targets only professionals in a narrow domain, limited learnability is less of a concern.

High efficiency is an expected strength of the concept. Product owners no longer have to write notes during meetings so they can create digital issues later. They are able to dictate the issue and the proposed application will create it automatically. This very efficient interaction with the system requires that the user is sufficiently trained and accustomed to the functionality. For power users who regularly use the application and know the features of the voice interface this can be a very efficient way to interact with a system.

The definition of the usability attribute errors is not quite sufficient for the use here. Nielsen references user errors and catastrophic system failures. An application that uses machine learning like the proposed concept here can also create errors in its decision making. For example, the name of an assignee in a spoken issue could wrongly be classified as a label for the issue. This is not a user error and also no catastrophic error. This error was caused by a wrong decision in the machine learning components. The error rate of these particular errors can be improved by further training of the machine learning models that are used.

User satisfaction is a more subjective usability attribute. For this concept, high user satisfaction can be expected. The application automates the previously manual task of creating new issues in the issue management system and aims to save the user time through high efficiency.

To summarize this evaluation: the concept of a voice assistant for issue creation focuses primarily on high efficiency. Other usability attributes like learnability and memorability are weaknesses in this design. However, when designing the application for a specific target user group of heavy users these tradeoffs are justifiable. For this intended user group, the poor learnability and memorability can be acceptable as long as the efficiency gains and user satisfaction are high enough.

4.2.3 Example Use Cases for Recognized Issues

The following section shows exemplary use cases that can be supported by this system. First the spoken text is shown. This is the text that a user of the system would speak into a microphone and that will be transcribed by the speech recognition service. After that the individual elements the natural language processing (NLP) engine would recognize are listed. From these recognized elements a structured issue is constructed.

Change Request

To create a new issue for a change request the product owner Peter would say:

[1] https://assistant.google.com/
[2] https://developer.amazon.com/alexa
[3] https://www.apple.com/siri/

Add a monitoring system to our server. The administrators should receive notifications if we have issues. This is for components payment and auth-service. Assign Fabio, add labels enhancement. The weight is 7 and the priority is high.

The application should then recognice the following elements:

- Title: “Add SSO Support”
- Issue Body: “The administrators should receive notifications if we have issues.”
- Cross-Component Issue for the following services: “payment-service”, “authentication-service”
- Lables: “enhancement”
- Priority: “High”
- Weight: “7”
- Assignee: the project member with user ID: “FabioSchmidberger”

Bug Report

To create a new bug report the developer Jake would say:

The login button is hidden on the mobile page. Add the labels bug and mobile. The components are frontend. Assign Jake.

The application should then recognize the following elements:

- Title: “The login button is hidden on the mobile page”
- Cross-Component Issue for the following services: “app-frontend”
- Lables: “bug”, “mobile”
- Assignee: the project member with user ID: “JakeCoder”

Only the issue title is a required element. All other issue elements can be left empty.

4.3 Architecture

The application will be created using a microservice architecture. This has the significant advantage of allowing individual services to be swapped and updated without impacting the overall functionality of the application. This is especially important for this thesis as it allows us to easily change the speech recognition service to use different speech recognition frameworks (like Kaldi, DeepSpeech, wav2letter++, or the Web Speech API). These frameworks also need specific environments to run in, so containerizing them using Docker is the best approach for reliable and repeatable deployment. This also allows each service to be built in the tech stack with the best framework support.

Figure 4.3 visualizes the architecture. In the following list the purpose of each service is explained:

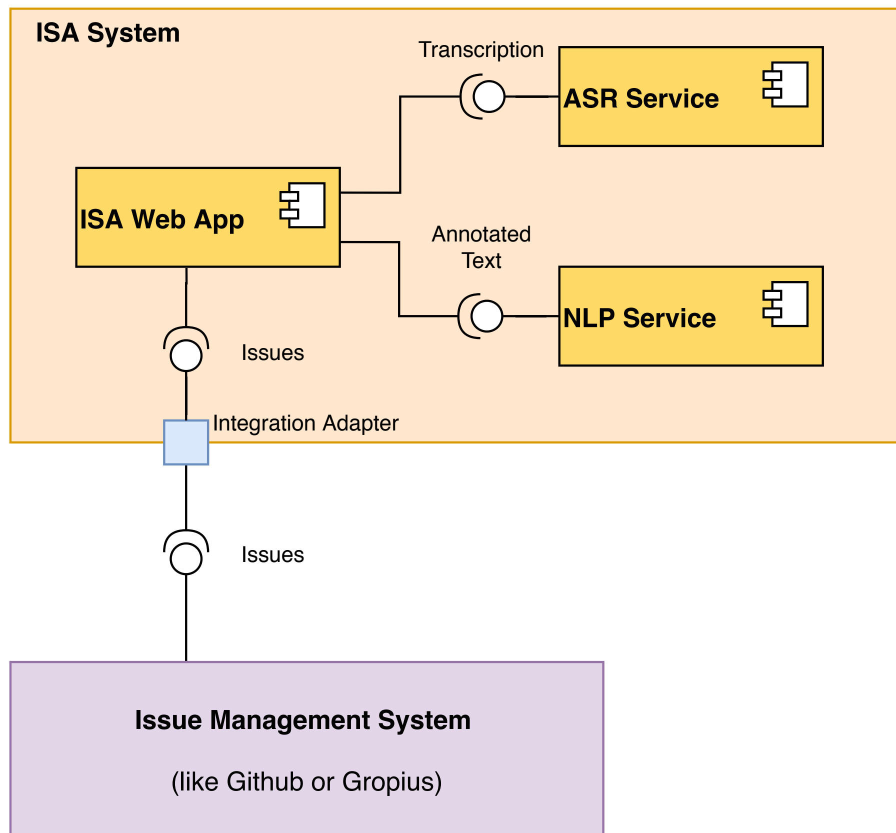


Figure 4.3: Service Architecture as a UML Component Diagram

1. **ISA Web App:** This ISA web app is the heart of the system. It displays the UI to the user and records the spoken audio input. The recorded audio is passed on to the ASR Service. The resulting transcription from the ASR service is then displayed and passed into the NLP service for annotation. Based on the transcription and annotation results a structured issue is created and displayed. The ASR service based on settings entered by the user, the integration adapter will be configured.

2. **ASR Service:** The ASR (automatic speech recognition) service contains the speech recognition engine and has the task to convert an audio input stream into a text output. A web socket connection will be used for continuous audio streaming and transcription.

3. **NLP Service:** This service contains CoreNLP and an API service with exposes the specific CoreNLP features needed in this application. The NLP Service annotates the text that is passed into it. A custom NLP result interface will be used by the NLP Service and the ISA web app to abstract the CoreNLP specific data structures.

4. **Issue Management System:** The issue management system is an external application that provides an API to create new issues. Multiple different issue management systems can be supported. As part of this thesis support for Github and Gropius will be implemented. The integration adapter uniforms the integration implementations under a uniform interface.

4.4 Speech Recognition Pipeline

In this thesis, an existing open-source speech recognition system will be used. Prominent options are Kaldi [Kal] and DeepSpeech [moz]. The newly released wav2letter++ [res] from Facebook research could also be a great option. The architectural approach outlined above will make it possible to easily exchange the speech recognition system for a new one or customize an existing speech recognition system to fit specific needs.

4.4.1 Language Support

The ISA system will only support English. This makes the ISA system available to the largest potential audience. As many software development teams write their issues in English even if they live in a non-English speaking country, the majority of developers should be able to use the ISA system.

Support for other languages like German could be added later. The modular architecture of the system allows other developers to replace the used speech recognition system and natural language processing with a different version that supports languages other than English.

However, adding multiple locales to the ISA application is considerably more work than in regular web applications. Support for new languages has to be added to the application UI, speech recognition, and the natural language processing system. Adding a new language to the natural language processing system is particularly time-consuming, as a new model has to be trained on a new data set. The current model is trained based on English example issues. To train the model in German, a German data set would have to be created.

4.4.2 From Spoken Input to Structured Issue

The goal of this thesis is to build a system that can automatically create structured cross-component issue entries from spoken text. A multi-step pipeline will be used to generate the structured input.

Figure 4.4 visualizes this pipeline. This pipeline is split into three phases:

1. Speech Recognition
2. Natural Language Processing
3. Manual Issue Confirmation and Synchronization

Phase 1: Speech Recognition

The input to the speech recognition phase is the spoken issue from the user. The spoken input of the user is captured by a microphone, which creates an audio stream. This audio stream is passed into the speech recognition toolkit.

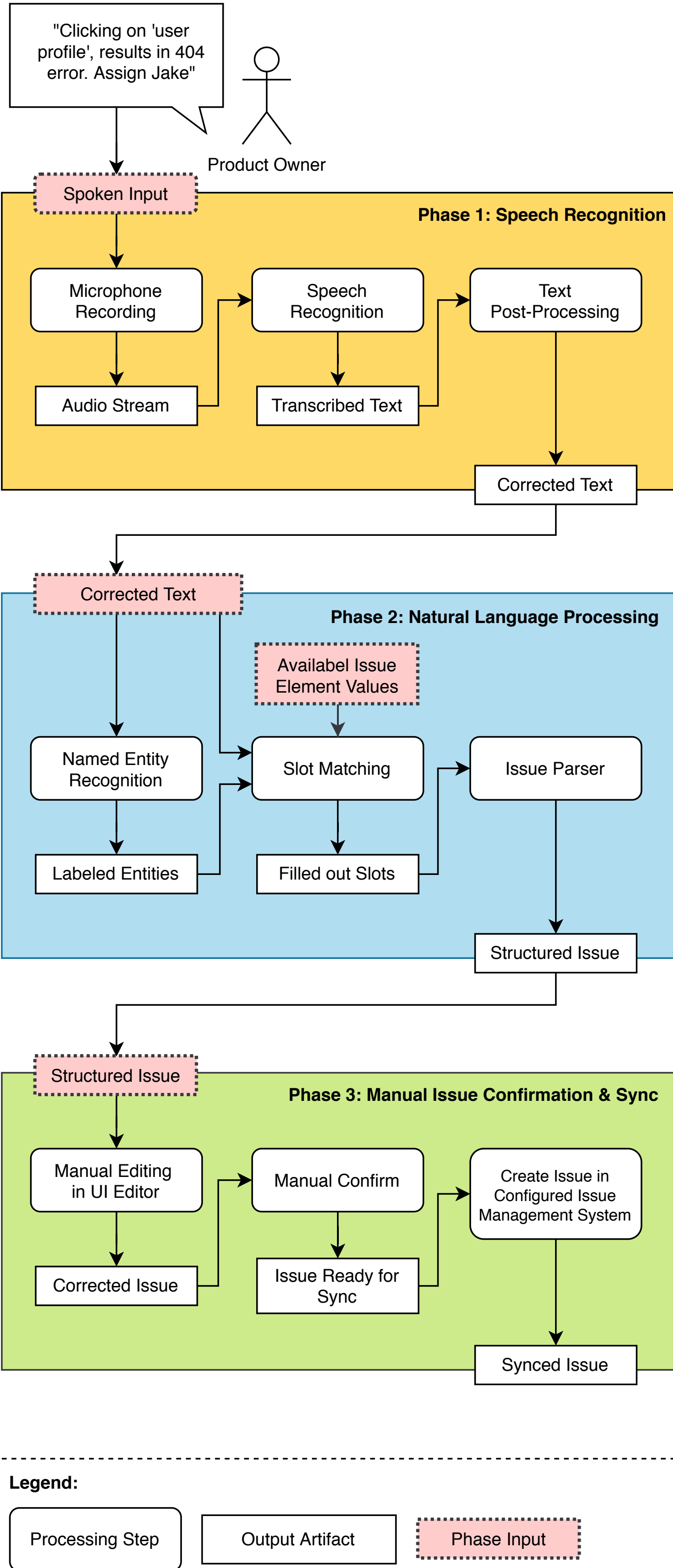


Figure 4.4: Speech Pipeline

The speech recognition toolkit returns `partial results` of the spoken text. These `partial results` are the text that the speech recognition has detected based on the audio stream it has received so far. The `transcribed text` is returned once the recording has ended and the speech recognition has transcribed the full audio input.

The `transcribed text` is then passed into a text post-processing step. In the post-processing, two kinds of corrections are made: text content correction and text formatting corrections. The text content correction uses heuristics to fix common mistakes made in the speech recognition step. For example, "assign" is often not recognized correctly by the speech recognition and written as "A sign". Mistakes like this are detected and get replaced in the text content correction by a regex replacer. The text formatting corrections include punctuation, word spacing, and capitalization. These text formatting corrections are executed after the text content correction.

The output of the speech recognition phase is a corrected `string` of the spoken input from the user. This string is called `Corrected Text` and gets passed on to the next phase.

Phase 2: Natural Language Processing

In the natural language processing phase, the resulting `Corrected Text` from phase 1 is passed in as an input. The text is sent to the named entity recognition microservice. This service analyses the text and recognizes the entities that are in the text. A list of the labeled entities is returned.

The step that takes the generated data and creates a structured element of the issue (like a list of labels or the title) from that data, will be called slot matching in this thesis. For the slot matching step now combines the labeled entities, the corrected text, and the available issue element values to generate the issue slot values. These slots are issue title, issue body, components, assignee, labels, issue weight, and priority. For slots like assignee, components, and labels there is a predefined selection of values configured in the issue management system. In the case of the assignee slot, there might only be the users "Jake Taper" and "Max Maier" defined as possible assignees in the issue management system. When the product owner says "assign Jake" the recognized name "Jake" has to be matched to the user "Jake Taper" and his corresponding user id. When the speech recognition makes an error and recognizes the name "Jimmy" rather than the spoken input "Jake" the correct value should still be selected. The slot matching uses a trigram search to find the best matching available values based on the recognized input.

The following explains the process of slot matching in greater detail with the example of matching a label element.

1. Extraction of recognized elements: based on the NER results from the CoreNLP microservice all occurrences of the entity `LABEL` are selected. This list represents all of the labels that were detected by the NER. As the total number of detected entities in a NER result is very limited (usually less than 20) a simple linear search is used.
2. Importing of available elements: Labels are predefined in the issue management system and these values are imported into the ISA system.

3. Matching of recognized elements to available elements: The string similarity of a recognized element to each of the available elements is compared. The available element with the highest similarity to the recognized element is picked and will be the value of the slot. This matching is crucial, as the speech recognition can make mistakes.

The matched slots are then assembled into a structured issue by the `isuse parser`. The issue parser returns a structured issue which is the final result of the natural language processing phase.

Phase 3: Manual Issue Confirmation and Synchronization

In the third phase, the issue is manually reviewed by the user and then synced with an existing issue management system. First, the user checks the automatically created issue from phase 2 for mistakes. The UI editor allows the user to correct any mistakes that might have happened or to add data that he or she forgot to say when dictating into the system. The result of phase 2 is considered a suggestion that is automatically generated based on user input. The user still has the power to manually review and change the decisions made by the speech recognition service and natural language processing.

Once the user considers the issue correct, he or she can save it. The saved issue is automatically created in an issue management system over an API call. A local copy of the issue is not saved. The reasons for this design approach are discussed in Section 4.6.3. After phase three the issue has been synchronized to an existing issue management system. The user can now dictate a new issue.

4.5 Natural Language Processing

Once the spoken utterances are converted into written text by the speech recognition service, the natural language processing service will analyze the written text to detect the semantics. For this, the very popular natural language processing toolkit CoreNLP as described in [MSB+14] will be used. The Stanford CoreNLP Toolkit is an open-source natural language processing (NLP) toolkit based on the JVM.

In this thesis named entity recognition (NER) will be mainly used to detect the structured components in the entered text. Coreference or bootstrapped entity learning presented in [GM14] look very promising as well. The further optimization of the natural language processing system by using these tools is beyond the scope of this thesis and a short outlook on these techniques will be confined to the Future Work section.

4.5.1 Named Entity Recognition (NER)

To automatically detect specific elements in the issue, named entity recognition will be used. Named entity recognition is only one of the many tools provided in CoreNLP, but the main focus of this thesis.

The CoreNLP NER Classifier is based on arbitrary order linear chain Conditional Random Field (CRF) sequence models as described in [FGM05] with further details given in [Corb]. These sources also show that CoreNLP NER has a *NERCombinerAnnotator*. This allows the use multiple different annotators, such as the *RegexNER* and *entitymentions* annotators. Their results are then combined by the *NERCombinerAnnotator*.

The *docdate, sutime, regexner, tokensregex, entitymentions* annotators are run as sub-annotators of the NER annotator as described in [Corb].

The CoreNLP NER Tagger handles sparse data sets exceptionally well. Training the NER Tagger with only 10 labeled example sentences per named entity (for 20 named entities total) already performed well. For this system approximately 10 named entities have to be recognized (see Section 4.5.3 for details). This means that manually creating labeled test data for 20 - 40 issue examples should be sufficient for acceptable accuracy.

These results can be further improved by utilizing the RegexNER annotator [Cora]. Considering the semi-structured user input when they dictate a issue they want to create, utilizing the RegexNER Annotator could be very promising. By using a couple of heuristics for the structured elements in an issue the recognition accuracy could be improved and the training significantly simplified.

This shows the great strength of the highly customizable NER Pipeline in CoreNLP and makes CoreNLP the ideal NLP tool kit for this system.

4.5.2 Training the Model

One of the challenges in this thesis will be to train a sufficient NER model to recognize the different entities in a Cross-Component Issue. As stated in Section 1.2 the purpose of this thesis is to build a proof-of-concept system. Therefore a basic NER model is sufficient and further extensive research in model optimization can be done in future work.

To train the NER tagger of CoreNLP labeled training and test data will be created manually. Examples of issue texts that users could say are labeled. Crawling existing open-source issues from sources like Github or Gitlab did not create the desired results. As these issues are already in a structured format and have very little similarity to the freely spoken input of a user, the crawled issues were not used to train the model.

4.5.3 Namend Entities in Cross-Component Issues

The following elements of an issue should be detected:

- Title
- Issue Body
- Components
- Assignee
- Priority

- Weight
- Lables

A particular challenge will be to accurately detect the *title* and *issue body*. These elements will have to be generated from the spoken text based on recognized named entities and heuristics about their structure. The first sentence will always be used as the issue title. If there is no recognized entity in the second sentence it will be used as the issue body. These heuristics are relatively basic. However, the use of NLP zoning to accurately extract title and body without relying on syntax is beyond the scope of this thesis. This approach could be evaluated in future work.

4.6 Issue Management System Integration

Once the structured issue has been created, it has to be transferred into existing issue management systems. This application should be decoupled from the issue management system it is integrated with. To achieve this service will be built, which provides its own so-called micro-frontend which is then displayed as part of the main issue management systems UI.

In this thesis, the ISA application will be integrated into the Gropius Cross-Component Issue Management System developed by [Spe19]. To allow support for additional issue management systems like Github and Gitlab, an adapter pattern will be used. This makes it easy to implement additional integration providers for these additional issue management systems.

4.6.1 Github Integration

Github provides a REST API[4] that allows users to access and create resources on Github, including issues and labels. The API is well documented. For the use in JavaScript there is a special npm package called `octokit`.

4.6.2 Gropius Integration

The Gropius cross-component issue management system also has an API that allows the ISA application to integrate into it. The code of the backend API is maintained in a public Github repository[5].

The Gropius Backend API uses GraphQL[6] instead of REST used by the Github API. A schema of the API can be generated from the code.

[4]https://docs.github.com/en/free-pro-team@latest/rest
[5]https://github.com/ccims/ccims-backend-gql
[6]https://graphql.org/

4.6.3 Persistance Design - Single Source of Truth

The ISA application is designed for issue creation and is not intended to be used as a fully-featured issue management system. As discussed in Section 2.1.2 issues are part of a whole issue management lifecycle that has multiple steps. The ISA application however is only active in a small part of the issue lifecycle. Therefore the data that is crucial for the issue lifecycle should be persisted and managed by the system that is responsible for the whole issue lifecycle any not by ISA.

It is important to ensure that there will be only a single source of truth for all issue management-related data. Two data sources (one in the issue management system and one in ISA) would result in the need for continuous synchronization and data duplication between the two systems. The goal was to avoid this to reduce complexity and the potential of errors.

For this reason, it was decided that the ISA application will not have its own persistence layer to store issues. ISA will exclusively rely on API integration into existing issue management systems to import context data from them and create new issues.

Not having a persistence layer has a number of benefits for the application design and greatly reduces the complexity both during development and during operations of the deployed application. As there is no stateful container the application can be scaled without the need for a distributed database. There is also no need for data backups or database migrations.

The ISA application has some configuration options that should be persisted. Rather than saving this configuration in a database for every user the data will be saved in the local browser storage provided by the Web Storage API[7]. This eliminates the need for user accounts and authentication but has the drawback that

[7] https://developer.mozilla.org/de/docs/Web/API/Web_Storage _API

5 Implementation

In this chapter, the concept presented in Chapter 4 will be implemented. The code written in this thesis is open-sourced and available on Github[1].

In Section 5.1 the implementation of the ISA web application is presented. This section shows UI screenshots to help the reader imagine how the user will interact with the application. The ISA application discussed in Section 5.1 uses a speech recognition system and natural language processing microservice.

In Section 5.2 the speech recognition system will implemented and the challenges that were encountered during the implementations are discussed. After this, the implementation of the natural language processing microservice will be presented in Section 5.3. Section 5.4 shows how the application deployment was implemented and explains the necessity of a public deployment of the ISA system.

5.1 Application Frontend

This section outlines the implementation of the web application that was built for the ISA issue speech recognition system.

The user interface is really important in this application. It has the purpose to transparently display the results from the speech recognition and natural language processing and allow for easy correction and editing of the resulting structured issue.

This allows the user to evaluate the results and recognize any errors that might have happened during the speech recognition. Unfortunately, the current state of the art speech recognition and natural language processing systems are not perfect and from experience still make frequent mistakes. The design approach used here, embraces the potential of errors and aims to build an application that allows users to easily identify and correct any potential errors before they confirm and save the generated issue.

5.1.1 Technology

The frontend application was build using written in TypeScript[2]. TypeScript is an extension of JavaScript with types. Type-checking helps during development, adds valuable documentation into the code and allows you to validate your code before running it. During build time the TypeScript code is then transformed into JavaScript by the TypeScript compiler.

[1] https://github.com/FabioSchmidberger/issue-speech-recognition
[2] https://www.typescriptlang.org/

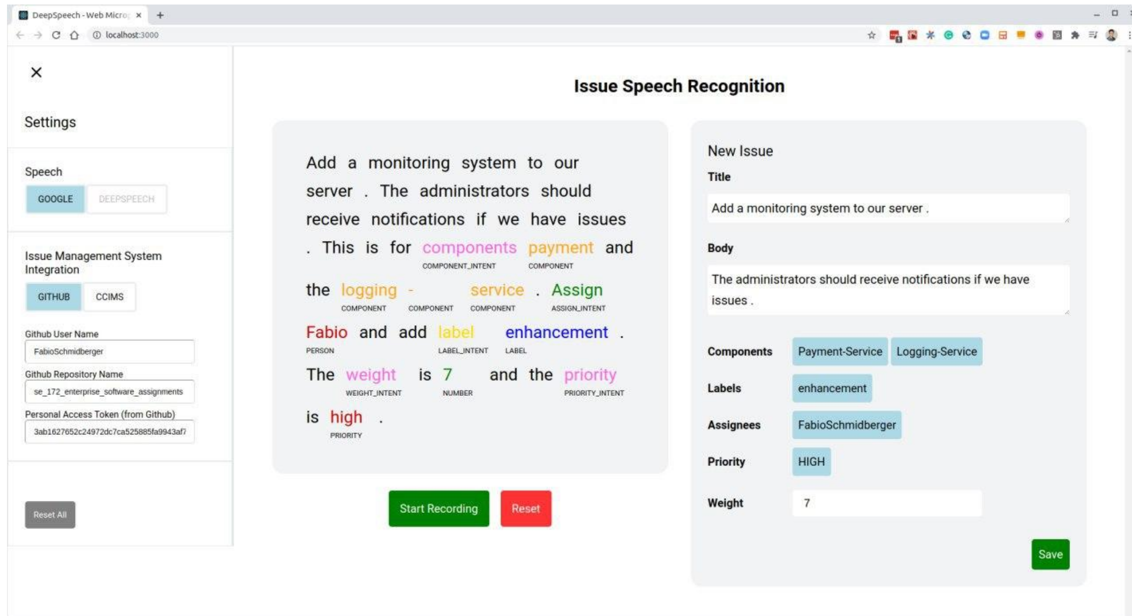


Figure 5.1: User Interface of the ISA application with speech input, issue card and collapsable settings

The Visual Studio Code[3] IDE was used for TypeScript development and has excellent language support.

React[4] was used as the UI framework. React is a JavaScript library that allows you to develop component-based user interfaces. This component-based design allows you to encapsulate the complexity of individual elements of your UI into separate components that can easily be reused.

5.1.2 UI Design

As described in Section 4.2, the application UI has two prominent elements: the speech input and the issue card. The speech input displays the spoken user input and the recognized elements from the text analysis. The issue card displays the structured issue that was created based on the speech input and allows the user to edit and confirm the issue. The user interface is shown in Figure 5.1.

In the following subsections, the individual components of the user interface are discussed in more detail.

[3]https://code.visualstudio.com/
[4]https://reactjs.org/

Figure 5.2: User Interface of the Speech Input Component

Speech Input

The speech input UI component allows the user to start and stop the audio recording and displays the recognized text. Based on the results of the natural language processing the detected entities are highlighted. To be transparent to the user, ISA color codes the recognized entities and adds a label of the entity name. Figure 5.2 shows the speech input UI.

Issue Card

The issue card allows the user to review, edit, and save the generated issue. The issue card can be seen in Figure 5.3.

The text content of the title and body can be edited simply by clicking in the text field and changing the text. This works on desktop and mobile. Issue elements like labels, components, and assignees can be changed by clicking on the elements.

When the user clicks the save button, the issue is passed to the integration adapter which will then use the appropriate configuration to create the issue in the selected issue management system.

Settings

The settings menu allows the user to select between the speech recognition systems that should be used. You can choose between `Google` and `DeepSpeech`. The `Google` selection uses the `Web Speech API` implementation in the Google Chrome browser as described in Section 5.2.2. When the `DeepSpeech` speech recognition service is used as described in Section 5.2.1.

The user can also configure the issue management system integration. For the Github integration a username, the repository name, and a personal access token is needed, to create an issue for the configured repository over the Github API.

Mobile Optimized User Interface

The ISA application was originally intended for use on a laptop or PC. However, the web application is also very useful as a tool on a smartphone. This allows users to speak into their smartphones to create issues.

Figure 5.3: User Interface of the Issue Card Component

The added benefit of the smartphone-optimized version is that the microphone on a smartphone is generally a lot better than the microphone integrated into many laptops. This results in improved speech recognition accuracy when using ISA on a smartphone.

To get an app-like feeling the web-application was implemented as a Progressive Web Application (PWA)[5]. PWAs allow users to add your web application to the home screen just like regular apps that were installed through the AppStore or Google Play Store. To create a PWA you have to add a `Service Worker`[6] and `Web Manifest` to the application. The service worker is able to cache application resources and can allow for offline capability of select functions of a web application. As the ISA application requires internet connection to use the speech recognition and create issues in existing issue management systems, the ISA PWA is not offline capable.

With a PWA a developer can create an application that can be added to the home screen like a native app, that support app notifications, and that can be offline capable. A huge advantage compared to regular native apps is that no approval from the Google PlayStore[7] or from the Apple AppStore[8] is

[5] https://web.dev/what-are-pwas/
[6] https://developers.google.com/web/fundamentals/primers/service-workers
[7] https://developer.android.com/distribute/best-practices/launch/launch-checklist
[8] https://developer.apple.com/app-store/review/

Listing 5.1 Redux Dispatch Event

```
const persistConfig = {
  key: 'root',
  storage,
  migrate,
  whitelist: ['settings'],
};

const persistedReducer = persistReducer(persistConfig, rootReducer);
```

Listing 5.2 Redux Dispatch Event

```
import { useDispatch } from 'react-redux';

...
const dispatch = useDispatch();

dispatch({ type: 'SET_LABELS', labels: labels })
```

required. These reviews can take up to two weeks and make it a lot more difficult to quickly release a bug fix. With a PWA a developer release an app update by simply pushing a new version to your server, no approval process is required. The service worker can be configured by a developer to automatically load the new version.

5.1.3 State Managment

Redux[9] was used as the state management framework. As described in Section 4.6.3 no database will be used to persist the application configuration. Instead the local browser storage will be used. This means that the data configured on one device will not be shared with any other device. This can have a negative impact on the user experience if a product owner user multiple different devices. However, this is justifiable considering the significant reduction in application complexity that is gained by not using a database.

The Redux state management makes it really easy to persist state in the local browser storage. A developer can use the `persistReducer` from `redux-persist` and whitelist which reducers should be persist to the browser storage. When the application is opened again from the same browser the saved state is loaded from the local browser storage. This works even after the browser was closed entirely or after the PC was restarted.

The Redux state management is event-driven. As shown in Listing 5.2, one can `dispatch` an event with the data that should be updated. This allows for great decoupling of your user interface code and the state management.

[9] https://redux.js.org/

Listing 5.3 Redux Reducer

```
function IssueElementsReducer(
  state: State = initialState,
  reduxAction: ReduxAnyAction,
) {
  switch (reduxAction.type) {
    case 'SET_LABELS':
      return {
        ...state,
        elements: {
          ...state.elements,
          labels: reduxAction.labels,
        },
      };
    default:
      return state;
  }
}
```

Redux Reducer listen on these events and if the `type` matches an action `type` defined in the switch statement of the reducer the managed state can be updated according to the event. Listing 5.3 shows a very simple example reducer that listens on the `SET_LABELS` event that was dispatched in Listing 5.2. Multiple reducers could also listen to the same dispatched events. Redux supports custom middleware, which makes it easy to add logging or user-interaction monitoring based on events.

5.2 Speech Recognition

This section describes the implementation of the speech recognition system used by ISA. First, the open-source speech recognition system DeepSpeech was used. The implementation of the DeepSpeech microservice is discussed in Section 5.2.1. The problems that were encountered with the DeepSpeech speech recognition system are explained here as well. These problems lead to the decision to use the Web Speech API instead of DeepSpeech. The implementation and usage of the Web Speech API for the speech recognition of the ISA application is discussed in Section 5.2.2.

5.2.1 DeepSpeech

DeepSpeech is an open-source speech recognition framework and was presented in greater detail in greater detail in Section 2.2.1. In the following, the implementation of the DeepSpeech Microservice and problems that were encountered with the DeepSpeech system are outlined.

DeepSpeech Microservice

The DeepSpeech speech recognition system was extracted into a separate microservice to allow for easy replacement of it should a new state of the art speech recognition system yield better results. Even during the writing of this thesis, this decision proved to be valuable.

The DeepSpeech Microservice was implemented in NodeJS[10]. The goal was to immediately display the recognized transcription even while the user is still speaking. This required an implementation based on WebSockets so that the audio data could be continuously streamed from the client's device to the DeepSpeech microservice hosted on a server.

DeepSpeech has great documentation and offers multiple examples of different use cases. One of these examples showed how DeepSpeech could be used in a NodeJS deployment. For the implementation of the DeepSpeech microservice, the example provided by the Mozilla Foundation was used as a reference and modified.

Problems with DeepSpeech

After the DeepSpeech microservice was implemented and integrated into the web application described in Section 5.1 basic tests were conducted to evaluate how well the DeepSpeech speech recognition worked for the ISA use case.

This evaluation showed significant accuracy problems when dictating issues into a microphone. Basic words like "assignee" or their names were not properly recognized. In the future, these accuracy problems could be eliminated by training a custom DeepSpeech speech recognition model based on collected training data. With a custom model, the recognized vocabulary could be optimized for issues.

For practical use in ISA DeepSpeech was not sufficient. Even basic issue texts like "The login button should be green instead of blue. Assign Max. The priority is low." were recognized with five or more mistakes. A formal study of the word error rate in this application configuration was not conducted. The general accuracy of the system will be evaluated in Chapter 6.

5.2.2 Web Speech API

Due to the accuracy problems encountered with DeepSpeech, it was decided to use the Google implementation of the Web Speech API instead. The theory behind the Web Speech API was described in Section 2.2.1 and this section will show the implementation details.

There is a special npm package that can be used for the Web Speech API in a React environment called `react-speech-recognition`[11]. This package makes it easy to use the Web Speech API in a react app and has support for the modern React Hooks. Listing 5.4 shows how the `useSpeechRecognition` Hook can be used to retrieve the transcript.

[10] https://nodejs.org/en/

[11] https://www.npmjs.com/package/react-speech-recognition

Listing 5.4 Web Speech API Consume Transcript

```
import SpeechRecognition, { useSpeechRecognition } from 'react-speech-recognition'

const { transcript, resetTranscript, listening } = useSpeechRecognition();

const options = { continuous: true, language: 'en-US' };

const startRecording = () => {
  SpeechRecognition.startListening(options);
};

const stopRecording = () => {
  SpeechRecognition.stopListening();
};
```

Listing 5.5 Web Speech API Browser Compatibility Test

```
import SpeechRecognition from 'react-speech-recognition'

if (!SpeechRecognition.browserSupportsSpeechRecognition()) {
  // Web Speech API is not supported
}
```

Browser Support

As described in Section 2.2.1 the Web Speech API currently has limited browser support. The code shown in Listing 5.5 allows a developer to verify if the current browser of a user supports the Web Speech API.

5.3 Natural Language Processing

In this section, the implementation of the natural language processing microservice is outlined. First, the usage of the CoreNLP annotation pipeline is shown. Furthermore, the implementation of the `HttpServelet` that serves the API of the microservice is presented. After this the model training process and creation of the training data is explained.

5.3.1 CoreNLP

Java was chosen as the language for the CoreNLP microservice because CoreNLP itself is written in Java and the CoreNLP Java SDK has good documentation. To add CoreNLP to the project it was included as a Maven dependency as shown in Listing 5.6.

To annotate text with CoreNLP an annotation pipeline has to be created by calling the `StanfordCoreNLP` constructor. The pipeline can be configured through props. This allows a developer to set which annotators should be used and configure the path to the model files. Details on how to

Listing 5.6 CoreNLP Maven dependency

```
<dependency>
  <groupId>edu.stanford.nlp</groupId>
  <artifactId>stanford-corenlp</artifactId>
  <version>4.0.0</version>
</dependency>
<dependency>
  <groupId>edu.stanford.nlp</groupId>
  <artifactId>stanford-corenlp</artifactId>
  <version>4.0.0</version>
<classifier>models</classifier>
```

Listing 5.7 CoreNLP Document Annotation

```
// build pipeline
StanfordCoreNLP pipeline = new StanfordCoreNLP(props);
// create a document object
CoreDocument document = new CoreDocument(text);
// annnotate the document
pipeline.annotate(document);
```

train a CoreNLP model are discussed in Section 5.3.3. A `CoreDocument` is created from the `text` that should be annotated. The `text` is of type `String`. This `CoreDocument` can then be annotated by the pipeline.

Listing 5.7 shows how the StanfordCoreNLP tool can be used to annotate a text.

5.3.2 Java Servelet

The CoreNLP microservice exposes a REST API that allows the API consumer to pass in a text that should be annotated and returns the annotated result. This API was implemented as a `HttpServelet`.

The REST API consists of a single `GET` endpoint on the `/corenlp` route of the Servelet. The consumer calls the endpoint with the `text` as query parameter. A JSON response with the annotation results in the body is returned. CORS Headers also had to be set by the CoreNLP microservice to allow web application that are using HTTPS like the ISA application to access the CoreNLP microservice.

The CoreNLP microservice was dockerized to run it locally and so it can be deployed later. As Tomcat will be used as a web server the `tomcat:jdk8-openjdk` base image was used. The generated `war` file from the Maven Build steps was moved to the Tomcat Webapps directory. The `web.xml` file was configured to serve the CoreNLP Servelet under the route `/corenlp`. The trained CoreNLP model files were also packaged into the container. The container will serve the application on port `8080`.

Listing 5.8 CoreNLP Training Data Example

```
add O
ios O
support O
for O
nlp O
assign ASSIGN_INTENT
Max PERSON
issue O
weight WEIGHT_INTENT
is O
4 NUMBER
labels LABEL_INTENT
are O
app LABEL
and O
feature LABEL

redesign O
admin O
frontend O
labels LABEL_INTENT
are O
app LABEL
weight WEIGHT_INTENT
is O
1 NUMBER
assign ASSIGN_INTENT
Jeff PERSON
```

5.3.3 Named Entity Recognition Training

To train the named entity recognition in CoreNLP labeled data has to be provided. Unfortunately, there is currently no pre-labeled data of naturally spoken issue text publicly available. This meant that the training data set had to be created manually as part of this thesis. To create a data set a developer has to write down spoken issue examples and label the entity of each word.

The data set is written in regular textfiles that represent a two-column table. The first element in each row is the word of the sentence and the second word in each row is the entity label of the word. Each word has to be on a separate line. Two sentences are separated by an empty line.

`Null-Entities` are words that in this configuration they are marked with an `O`. Named-Entities are indicated by labels that are not `O` such as the entities `PERSON`, `LABEL` or `NUMBER`. You can create a new entity simply by adding a previously unused label to the training data. There is no need to globally define all labels that will be used.

Listing 5.8 shows a small excerpt of the training data that was used to train the models of the CoreNLP microservice.

Listing 5.9 Command to train CoreNLP Model

```
# train model
java -Xmx2g -cp "$CORE_NLP_PATH" edu.stanford.nlp.ie.crf.CRFClassifier -prop ner.model.props
```

CoreNLP Training

Once the data set was created, it has to be split it into a training data set and a test data set. The training data set will be used to train the CoreNLP model and the test data set will be used to evaluate the performance of the newly trained model.

The `build-model.sh` bash script was created to automate the model creation. Listing 5.9 shows a small part of the script that is responsible for training the model. Based on the configuration set in the `ner.model.props` a model will be trained.

The output of this command is a `ner.model.ser.gz` file that contains the trained named entity recognition model. In the CoreNLP microservice, this model file is baked into the docker container and used during the document annotation.

5.4 Application Deployment

Easy deployment is important for this tool. It allows users and other developers to quickly deploy the application locally and try out the functionality. In this section, the reasons why deployment is required for this thesis will be outlined and details on the technical implementation of the deployment setup are given.

5.4.1 Necessity of deployment

In order to, conduct the experiment and get feedback from users safely and easily, the application had to be used remotely with no setup need. In-person meetings with users and an experiment that required an in-person session are currently not possible. For this reason, it was not an option to have the system run locally on a developer's laptop and let the study participants use this laptop to complete the experiment.

The application had to be deployed and be publically accessible through a browser on the personal laptop or PC of every study participant. While this deployment meant additional work and took a lot of time, it was necessary for the completion of this thesis.

5.4.2 Deployment Orchestrator Decision

The application will be containerized using Docker. For the orchestration of the multiple containers that are used in this application two tools would be suitable: docker-compose and Kubernetes.

Docker-compose is a basic orchestration tool. On their website [Doc] they describe it as "Compose is a tool for defining and running multi-container Docker applications."

Kubernetes is a far more advanced orchestration tool with a different focus. On the Kubernetes website the tool is described as: "Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.". The focus here is on scalability and being able to manage and orchestrate multiple different applications, where each of them can consist of multiple containers.

While docker-compose is meant to be used to run multi-container applications on a single host machine, Kubernetes is designed to allow you to distribute your containers multiple different host machines (so-called nodes). Kubernetes also allows for easy and fully automated deployment of multiple applications on the same cluster.

However, docker-compose has a low barrier to entry and the deployment setup is considerably easier. The advanced functionality with multi-node deployments and orchestration of multiple applications is not needed here. For a developer who wants to try out the application, it is far easier to get a 'docker-compose' setup running.

For its ease of use docker-compose was chosen as the deployment orchestrator in this thesis. Should a developer want to deploy the application on a Kubernetes cluster rather than with docker-compose they can use one of the tools like `Kompose`[12] which can automatically convert a docker-compose file to Kubernetes deployment configuration files.

5.4.3 Infrastructure Architecture

The application has four separate containers: the App container, the CoreNLP container, the DeepSpeech Container and the Reverse-Proxy container. The App container contains the react web-application, Nginx is used as the webserver in this container. The CoreNLP container provides the CoreNLP API and is implemented as a Java Servlet. In this case, a tomcat-webserver is used. The DeepSpeech service was implemented as a NodeJS application. To make these containers publically available and add HTTPS support a reverse-proxy was added. The reverse-proxy also uses Nginx. The deployment architecture is visualized in Figure 5.4

Host

For the host system, an ubuntu 20.04 VM was chosen. Docker was installed using the provided guide[13] from docker.

Currently, there is an issue with the default ssh configuration and docker-compose. When using docker-compose to deploy an application to a remote docker host, it is required that you increase the maximal number of concurrent allowed ssh connections on the docker host to at least 30 concurrent connections.

The host system with ubuntu 20.04 has no other configuration changes.

[12] https://kompose.io/

[13] https://docs.docker.com/engine/install/ubuntu/

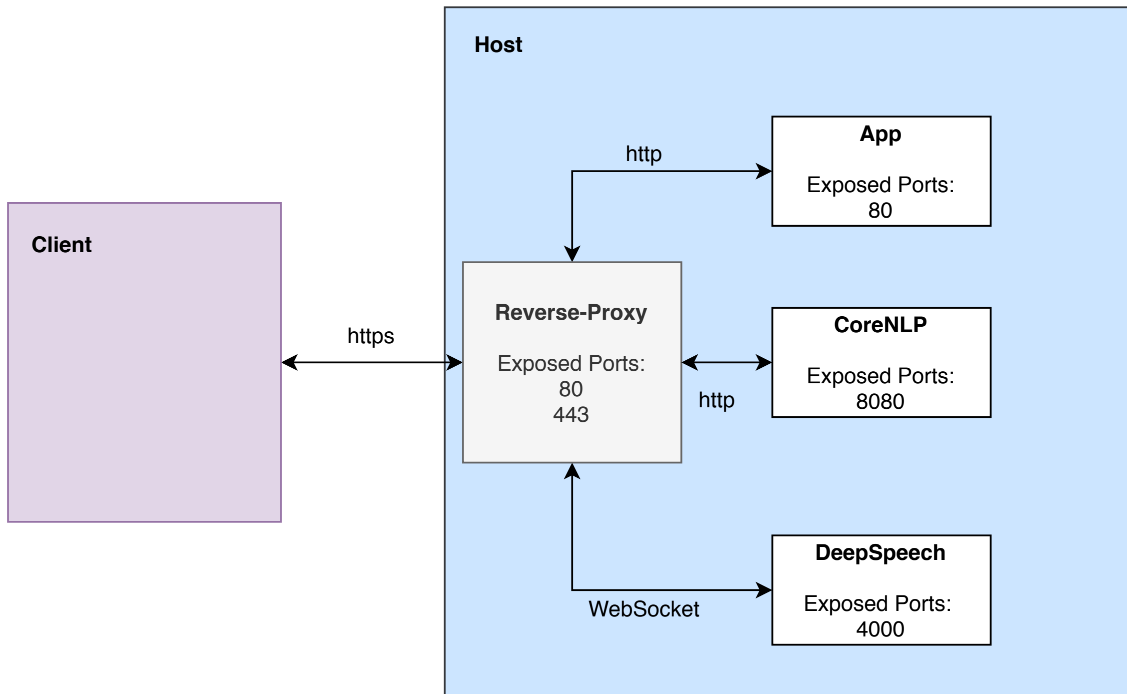


Figure 5.4: Application Deployment Architecture

Listing 5.10 Host SSH Configuration

```
echo "MaxSessions 500" >> /etc/ssh/sshd_config
```

Let's Encrypt Certificates

For the deployment of the application it was required to add HTTPS support. Adding HTTPS support to the deployed application was necessary, as the Chrome Browser only allows the microphone usage if the website uses HTTPS rather then unsecured HTTP.

In order to add HTTPS free SSL certificates from the Let's Encrypt[14] service where used. Let's Encrypt allows a developer to automatically generate free certificates.

Let's Encrypt provides the `certbot`[15] docker image. When creating a certificate for a domain the certificate authority (in this case Let's Encrypt) has to verify that you are the rightful owner of this particular domain and have control over it. For this deployment the DNS-01 challenge was used. With this challenge the `certbot` generates a token that has to be added as a `TXT` Record under `_acme-challenge.<YOUR_DOMAIN>` in the DNS configuration. Depending on the DNS provider of the domain it is possible to automate the certificate renewal process. Unfortunately, not all DNS providers have the required API support for this. The DNS provider Route 53 from Amazon Web Services supports automatic certificate creation and renewal. Many smaller DNS providers like

[14] https://letsencrypt.org/de/
[15] https://hub.docker.com/r/certbot/certbot/

Listing 5.11 nginx.conf - Reverse Proxy Configuration

```
server {
  listen 443 ssl http2;
  server_name  ${NLP_HOST_NAME};

  ssl_certificate /etc/letsencrypt/live/${SSL_CERTIFICATE_NAME}/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/${SSL_CERTIFICATE_NAME}/privkey.pem;

  location / {
      proxy_pass http://nlp:8080/;
  }
}
```

the German service Strato[16] have limited support for automated certificate renewal. Let's Encrypt certificates are valid for up to 90 days. If the DNS provider does not have API support it is required to manually update the certificate. To do so, a simple `certbot` command has to be executed to refresh the certificate.

Reverse Proxy

The reverse proxy handles the incoming traffic and decrypts it. A `Nginx` container is used as the reverse proxy. The decrypted traffic is then routed into the correct application depending on the URL that was used. In Nginx, this can be configured with the `server_name` attribute.

The docker-compose networking conveniently allows a developer to route traffic into a container by simply referencing the name of the container. With `http://nlp:8080/` for example you can access port `8080` on the container which is named `nlp` in the `docker-compose.yml` file. The simplified version of the reverse proxy configuration for the nlp host endpoint can be seen in Listing 5.11.

An automatic redirect to HTTPS for any incoming HTTP traffic was added as well. This ensures that all connections use the encrypted HTTPS.

5.4.4 Deployment Configuration

As described earlier docker-compose was used as the deployment orchestrator. When using docker-compose you create a `docker-compose.yml` file with your deployment configuration.

A big advantage of using a deployment orchestrator like docker-compose is, that you can configure containers to automatically restart in case the host machine reboots or even if the container crashes. This can be set with the `restart: unless-stopped` option. The full deployment configuration is shown in Listing 5.12

[16]https://strato.de/

Listing 5.12 docker-compose.yml deployment configuration

```
version: "3"

services:
  reverse-proxy:
    build: reverse-proxy
    restart: unless-stopped
    volumes:
      - certificates:/etc/letsencrypt
    ports:
      - "80:80"
      - "443:443"
    environment:
      - APP_HOST_NAME=${APP_HOST}
      - NLP_HOST_NAME=${NLP_HOST}
      - SSL_CERTIFICATE_NAME=${DOMAIN}
    depends_on:
      - app
      - nlp
  app:
    build:
      context: app
      args:
        REACT_APP_NLP_URL: https://${NLP_HOST}/api/corenlp
        REACT_APP_BASENAME: /
        PUBLIC_URL: https://${APP_HOST}
    restart: unless-stopped
  nlp:
    restart: unless-stopped
    build:
        context: nlp
    ports:
      - "8080:8080"

volumes:
  certificates:
```

6 Evaluation

In this chapter, the performance ISA system will be evaluated. An experiment was conducted for this evaluation. First, the experiment design is outlined in Section 6.1. In Section 6.2 the experiment results are presented and interpreted. To conclude the evaluation, potential threats to the validity of the experiment results are discussed in Section 6.3.

6.1 Experiment Design

An experiment will be used to evaluate the recognition accuracy of the ISA system. The experiment participants will receive example texts that they are instructed to read into the ISA system and record the resulting issue created by ISA.

It is necessary to use predefined issue texts so that the spoken text is consistent. Allowing the user to freely choose the phrasing, would rather result in an evaluation on how intuitive it is to speak a recognizable sentence and how accustomed the user is to interacting with digital voice assistants. However, this is not the objective of the evaluation. The goal is to measure how accurately ISA can create a structured issue given spoken input of different users. As discussed in Section 4.2.2 the ISA system inherently suffers from poor learnability. It takes a user time to get used to the digital voice assistant and learn what words and phrases have the best recognition accuracy most effectively given a certain task. As each participant was only able to spend a limited time with the application during the experiment, predefined texts were required.

Each study participant received 10 predefined issue texts that they read into the ISA application. The participants accessed the application through a web browser on their personal laptops. Prior to the experiment a basic questionnaire on the participant's demographic and background was filled out. 8 individuals participated in the experiment as shown in Table 6.1.

Participant ID	Age Group	Gender	Background	Native English speaker
1	20-30	male	software engineering student	No
2	20-30	male	software engineering student	No
3	20-30	male	software engineering student	No
4	20-30	male	product owner	Yes
5	20-30	female	(non-technical) Domain Expert	No
6	50-65	male	product owner	No
7	20-30	male	software engineering student	No
8	20-30	male	project management	No

Table 6.1: Experiment Participants

Precision	Recall	F1
0.945	0.77	0.853

Table 6.2: Precision, Recall and F1 Score of the ISA system

6.2 Results

For the evaluation of the experiment the precision, recall and F1 score of the ISA system will be evaluated. These are industry-standard metrics to evaluate a machine learning model.

The precision of a system is the number of true positives divided by the sum of true positives and false positives. The precision indicates what percentage of the labeled elements were correctly labeled. The precision score can be between 0 and 1. A precision score of 1.0 indicates that every element that is part of an entity group E was labeled as E by the system.

The recall is the number of true positives divided by the sum of true positives and false negatives. The recall indicates what percentage of elements that are supposed to be labeled got correctly labeled. The recall score can be between 0 and 1. A recall score of 1.0 indicates that all no element was missed.

The F1 score is the weighted average of precision and recall. It can be calculated with the following formula: $F1 = \frac{2*Recall*Precision}{Recall+Precision}$

In the experiment the true positives, the false positives, and the false negatives in the results of ISA were counted. Correctly labeled null-entities are not counted towards the true positives. From these measured values the precision, recall, and F1 score of the ISA system were calculated to determine the system performance. Table 6.2 shows the resulting values.

The F1 score is relatively high, indicating a good system performance. Overall the system performance ISA is respectable. This result is slightly worse than the pertained CoreNLP models which achieved an F1 score of 0.913[1]. The overall recognition of the ISA system has to be further improved in future work. When creating an issue it is rare (less than 20% likelihood) that no edits on the issue card have to be performed. Especially, the issue title and issue body are likely to contain some errors. The content of these elements is taken directly from the speech recognition system with limited post-processing. As discussed in Section 4.2.2 it is also not very intuitive and to get optimal results the user needs to know which labels, components, and assignees are part of the system. However, some participants mentioned that the small fixes on the auto-generated issue by ISA seem to be easier than creating the entire issue manually.

It should be noted that microphone performance can have a big impact on the speech recognition results and subsequently the recognized issue. The experiment participants mostly used webcam microphones or their smartphones.

These results show that ISA is a good first step to automatically recognize structured issues from spoken text. The speech recognition and natural language processing systems have to be further improved for this system to be a tool that can be daily used by software development teams.

[1] https://nlp.stanford.edu/software/crf-faq.html

6.3 Threats to Validity

This section discusses the threads to the validity of the concept developed in this thesis.

6.3.1 Construct Validity

The evaluation results are based on precision, recall, and F measure. Other evaluations might have different results. But these measures are widely used to evaluate NLP and NER systems.

As discussed during the experiment design in Section 6.1 the experiment only evaluated the recognition accuracy of the ISA system given a predefined text. A long term study of the concept in a more real-world situation, would allow for more robust results.

6.3.2 Internal Validity

The data set used to train the ISA system was based on manually created data. This data does not necessarily represent data that will be spoken into the system in a real-world situation accurately. Due to the lack of available production data, some assumptions on how the user would interact with it had to be made. As the data was manually created and labeled the volume of it is very limited. Following this thesis, a long-term field study of the ISA system would be required to give a more accurate representation of its real-world performance.

As the data was labeled manually human mistakes can not be avoided. The values of the true positives, false positives and false negatives in the experiment were also manually collected.

6.3.3 External Validity

The experiment participants were predominantly young male professionals with a background in software development. For most participants, English was their second language. The lack of diversity in the participant group could be a threat to the validity of the experiment results.

7 Conclusion and Future Work

This chapter summarizes the results of this thesis in Section 7.1. The key insights of the ISA concept are outlined. Finally, Section 7.2 presents opportunities for further research in the area.

7.1 Results and Conclusion

This thesis identified a problem with the issue creation process. Issues are an important tool to document change requests and bug reports. However, there is a time and location gap between the need to create an issue and the actual act of creating this issue in the issue management system. The need to create an issue often comes up during a team meeting, these issues however are often created hours after the meeting. As the issue creation in current issue management systems is too time-consuming to create issues during a meeting. This can negatively impact the quality of the issues as important details are forgotten or the user can forget to create the issue outright. This can cause the team to not plan and execute on the issue.

Interviews were conducted to learn more about the problem and gather requirements from potential users. Based on these requirements a new concept for a speech assistant that could automatically create structured issues from freely spoken text was developed. This system is called ISA, short for Issue Speech Assistant.

In the ISA system speech recognition is used in the first step to create a transcription of the spoken text. This text is then analyzed by natural language processing and named entities are extracted. Based on these results a structured issue card is filled out. The user can review and edit the generated issue. Once the user saves the issue it is transferred into an existing issue management system.

A prototype of the ISA system was implemented as part of this thesis. The code is available on Github[1]. A microservice architecture approach was used. For the speech recognition a DeepSpeech service was implemented. As the available DeepSpeech models had accuracy problems, support for the Web Speech API was added as well. The NLP microservice uses CoreNLP named entity recognition to annotate the text result. The architecture was designed with extensibility in mind and allows for easy replacement or extension of individual services.

This proof-of-concept implementation allowed for an evaluation of the ISA design. The goal of the evaluation was to determine if further research in this area can be justified. An experiment was conducted to measure the performance of the system. Based on the experiment results and user feedback, the concept design was validated and further optimizations will be implemented.

[1] https://github.com/FabioSchmidberger/issue-speech-recognition

ISA showed how a digital voice assistant can be optimized for a specific use case in a limited domain. By targeting professional power-users rather than general consumers the voice assistant is able to focus on efficient user interaction at the cost of intuitiveness. This concept of a domain-specific voice-centric system could be extended into other domains. This approach is in contrast to current well-known digital voice assistants like Goole Assistant, Alexa or Siri. However, the expected benefits in a professional context and the achievable system accuracy is higher than in the existing consumer-focused offerings. A careful analysis of the existing processes is required to make sure the speech assistant is optimally integrated into the user process and the application landscape the organization uses.

To summarize, the ISA system streamlines the issue creation process and makes it possible for product owners to easily create issues during a meeting. Spoken user input is automatically translated into a structured issue. The application implemented in this thesis, validated the developed concept and showed promising enough results to justify further research and analysis of additional use cases.

7.2 Future Research Opportunities

This section outlines areas where further research on the thesis topic could be conducted. First possible improvements to the speech recognition and natural language processing are discussed. A machine learning pipeline is proposed which would instrumental to achieving these improvements. Furthermore, a new design for the integration system is presented, that could make it easier to integrate into additional issue management systems. To conclude this section, additional use cases beyond the current prototype implementation of the ISA concept are presented.

7.2.1 Improved Speech Recognition

The speech recognition accuracy is still a weakness of the system. As this application has a limited domain (issue management) and therefore a limited vocabulary a custom language model and acoustic model could significantly increase the accuracy. Currently, the Web Speech API is used, which does not support custom language models or acoustic models. However, for DeepSpeech custom models could be trained.

7.2.2 Improved Natural Language Processing

In this thesis, a basic NLP system based on named entity recognition and heuristics was used. This approach has its limitations. Elements like the issue title and issue body are recognized with a heuristic and the recognized input text is simply pasted into the issue card. An improved semantic understanding of the spoken issue could improve the user experience and increase the resilience of the system to intuitively spoken text.

7.2.3 Machine Learning Pipeline

The previous sections described, how improved speech recognition and natural language processing is necessary for the ISA application. One of the challenges is, that there is very limited publically available training data that can be used to optimize the speech recognition and natural language processing for the domain of issue management. It would be a huge advantage if the ISA application would collect all the data that is created by using it. The audio from spoken input can be used to train the speech recognition, and based on the text results the NLP system could be trained.

A pipeline would have to be build where the data is collected, then labeled and stored in a new data set. This new data set can then be used to train new and improved models. It is very important to quality assure the incoming data to prevent wrongly labeled data from negatively impacting the model accuracy.

7.2.4 Integration Design

The ISA application currently only supports Github and Gropius as issue management systems. There are no integrations into other issue management systems at the moment. The used API by ISA is relatively slim, so it is not difficult to add additional issue management systems. But considering the vast number of different issue management and project management tools out on the market, an implementation of all their APIs is simply not realistic. A great way to have potential support for every tool is to build an integration to an integration platform like Zapier[2]. Zapier is an integration platform that allows users to configure adapters between over 2000 applications. Tools like Gitlab[3], Jira[4], Asana[5], and Github[6] all have Zapier support. Zapier does not require its users to write code and makes it simple to add integrations. Once a user has configured an integration between ISA and a new issue management tool, all other users will have access to it. Crowdsourcing integrations and empowering users to configure their own integrations without having to submit code to the repository would make it a lot more user friendly to add integrations.

7.2.5 Additional Use Cases

In the following, additional use cases for the ISA system are presented. The current implementation is a prototype to showcase the possibilities of the ISA concept. The following use cases could turn this concept into a viable product.

[2]https://zapier.com/platform
[3]https://zapier.com/apps/gitlab/integrations
[4]https://zapier.com/apps/jira-software/integrations
[5]https://zapier.com/apps/jira-software/integrations/asana
[6]https://zapier.com/apps/jira-software/integrations/github

Integration into Video Meeting or Remote Communication Tools

In the present climate, there is a significant increase in the number of meetings conducted remotely over video call software. The ISA tool developed for this thesis was intended to be used by a product manager during in-person meetings to easily create issues and make sure that all team members are on the same page.

But the tool could also be integrated into an existing video call platform like Zoom[7] or Cisco WebEx[8] and suggest the creation of a todo item or issue automatically based on the topics discussed. As the entire discussion is live recorded it is not very difficult to take the audio recording tracks and stream them into a speech-to-text system. The resulting transcript could be analyzed live during the meeting and suggestions for issues can be displayed directly in the video call software. This would allow the entire team to immediately see the generated issue and work together on it to clear any misunderstandings.

A similar approach could be taken with the Slack integration. When employees discuss something in a Slack channel the text could be automatically analyzed with the natural language processing system developed for this thesis and a suggestion to create an issue could be posted to the channel using a Slack bot.

Support for general Project Management Tools

This thesis focused on integrating ISA into issue management systems. The current implementation of ISA is very developer-focused, as the intended users are only product owners and developers. However, ISA could be very beneficial to other user groups like project managers. To extend in general project management the current data model of an issue (or rather task) needs to have customizable attributes. This is required, as most project management systems like Monday[9] and Asana[10] offer their users customizable data types. This would introduce additional challenges for the NLP system. Additional integrations into project management systems would be required.

[7]https://zoom.us/
[8]https://www.webex.com/de/video-conferencing.html
[9]https://monday.com/
[10]https://asana.com/

Bibliography

[ABD+19] R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, G. Weber. "Common voice: A massively-multilingual speech corpus". In: *arXiv preprint arXiv:1912.06670* (2019) (cit. on p. 14).

[AK17] C. Anantaram, S. K. Kopparapu. "Adapting general-purpose speech recognition engine output for domain-specific natural language question answering". In: *arXiv preprint arXiv:1710.06923* (2017) (cit. on p. 15).

[Bro19] N. Brousse. "The issue of monorepo and polyrepo in large enterprises". In: *Proceedings of the Conference Companion of the 3rd International Conference on Art, Science, and Engineering of Programming*. 2019, pp. 1–4 (cit. on p. 7).

[Cho20] K. Chowdhary. "Natural language processing". In: *Fundamentals of Artificial Intelligence*. Springer, 2020, pp. 603–649 (cit. on p. 11).

[CLZZ19] D. Chen, B. Li, C. Zhou, X. Zhu. "Automatically Identifying Bug Entities and Relations for Bug Analysis". In: *2019 IEEE 1st International Workshop on Intelligent Bug Fixing (IBF)*. IEEE. 2019, pp. 39–43 (cit. on p. 14).

[Cora] S. CoreNlp. *CoreNLP RegexNER Annotator*. https://stanfordnlp.github.io/CoreNLP/regexner.html (cit. on p. 30).

[Corb] S. CoreNlp. *Stanford Named Entity Recognition*. https://nlp.stanford.edu/software/CRF-NER.html (cit. on p. 30).

[Doc] Docker. *Docker Compose*. https://docs.docker.com/compose/ (cit. on p. 43).

[FGM05] J. R. Finkel, T. Grenager, C. Manning. "Incorporating non-local information into information extraction systems by gibbs sampling". In: *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2005, pp. 363–370 (cit. on p. 30).

[GM14] S. Gupta, C. D. Manning. "Improved Pattern Learning for Bootstrapped Entity Extraction". In: *Computational Natural Language Learning (CoNLL)*. 2014 (cit. on p. 29).

[HCC+14] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, et al. "Deep speech: Scaling up end-to-end speech recognition". In: *arXiv preprint arXiv:1412.5567* (2014) (cit. on p. 9).

[JJK+18] C. Jaspan, M. Jorde, A. Knight, C. Sadowski, E. K. Smith, C. Winter, E. Murphy-Hill. "Advantages and disadvantages of a monolithic repository: a case study at google". In: *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. 2018, pp. 225–234 (cit. on p. 7).

[Kal] Kaldi-asr. *Kaldi Speech Recognition Toolkit Github Repository*. https://github.com/kaldi-asr/kaldi (cit. on p. 26).

[LB] P. Laubheimer, R. Budiu. *Intelligent Assistants: Creepy, Childish, or a Tool? Users' Attitudes Toward Alexa, Google Assistant, and Siri*. https://www.nngroup.com/articles/voice-assistant-attitudes/ (cit. on p. 1).

[moz] mozilla. *DeepSpeech Github Repository*. https://github.com/mozilla/DeepSpeech (cit. on p. 26).

[MSB+14] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, D. McClosky. "The Stanford CoreNLP natural language processing toolkit". In: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 2014, pp. 55–60 (cit. on pp. 3, 11, 12, 29).

[Nie94] J. Nielsen. *Usability engineering*. Morgan Kaufmann, 1994 (cit. on p. 22).

[PGB+11] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, et al. "The Kaldi speech recognition toolkit". In: *IEEE 2011 workshop on automatic speech recognition and understanding*. CONF. IEEE Signal Processing Society. 2011 (cit. on p. 9).

[res] facebook research. *wav2letter++ Github Repository*. https://github.com/facebookresearch/wav2letter (cit. on p. 26).

[SBB20] S. Speth, U. Breitenbücher, S. Becker. "Gropius—A Tool for Managing Cross-component Issues". In: *European Conference on Software Architecture*. Springer. 2020, pp. 82–94 (cit. on pp. 2, 3, 7, 20).

[Spe19] S. Speth. "Issue management for multi-project, multi-team microservice architectures". MA thesis. 2019 (cit. on pp. 2, 5, 7, 31).

[VKWM15] M. Vogel, W. Kaisers, R. Wassmuth, E. Mayatepek. "Analysis of documentation speed using web-based medical speech recognition technology: randomized controlled trial". In: *Journal of medical Internet research* 17.11 (2015), e247 (cit. on p. 1).

[YD16] D. Yu, L. Deng. *AUTOMATIC SPEECH RECOGNITION.* Springer, 2016 (cit. on pp. 7, 8).

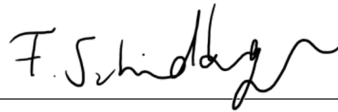
[YXF+16] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li, N. Kapre. "Software-specific named entity recognition in software engineering social content". In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 1. IEEE. 2016, pp. 90–101 (cit. on p. 14).

[ZLSG18] C. Zhou, B. Li, X. Sun, H. Guo. "Recognizing software bug-specific named entity in software bug repository". In: *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE. 2018, pp. 108–10811 (cit. on pp. 13, 14).

All links were last followed on Dezember 8, 2020.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Kornwestheim, 11.12.2020 

place, date, signature