RWTH AACHEN UNIVERSITY
Chair of Computer Science 5
Information Systems & Databases
Prof. Dr. Stefan Decker

Bachelor Thesis

Chatbot-assisted Community Analysis

Ben Aziz Lakhoune
Matr.-No.: 380163
Study Program: Bachelor Computer Science
May 5, 2021

Supervisors:	PD Dr. Ralf Klamma Chair of Information Systems RWTH Aachen University
	Prof. Dr. Matthias Jarke Chair of Information Systems RWTH Aachen University
Advisors:	Alexander Neumann Chair of Information Systems RWTH Aachen University

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Lakhoune, Ben Aziz	380163
Name, Vorname/Last Name, First Name	Matrikelnummer (freiwillige Angabe) Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende ~~Arbeit~~/Bachelorarbeit/ ~~Masterarbeit~~* mit dem Titel
I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

Chatbot-assisted Community Analysis

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Aachen, May 5, 2021	
Ort, Datum/City, Date	Unterschrift/Signature

*Nichtzutreffendes bitte streichen
*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Aachen, May 5, 2021	
Ort, Datum/City, Date	Unterschrift/Signature

Contents

List of Figures

1 Introduction

Community Information Systems (CIS) help Communities of Practice (CoPs) in structuring their work by offering tools to coordinate and evaluate different tasks within the community. A CoP consists of members with various degrees of knowledge and interest in the field. Researchers are the core members of the CoP. They contribute to the *success awareness* of the CoP with the help of so-called CIS *success models* [Klam10c]. Technical know-how is required to modify those success models, thus hindering the participation of less experienced members in the development process.

Meanwhile, chatbots are offering an intuitive interface to perform tasks with varying degrees of complexity. Therefore, the use of chatbots for success modeling could increase the *success awareness* non-core members, which increases the overall success awareness of the CoP.

1.1 Motivation

The rise of the World Wide Web allowed researchers to collaborate more easily. Researchers from all over the world can meet and exchange information more easily. A community of researchers studying a particular domain is referred to as (online) Community of Practice (CoP) [Renz08]. The success of CoPs relies on various internal and external factors, such as the active participation of its members or the general interest in the field. The evaluation of success is described as a dynamic process, as various needs of a CoP are changing over time [Renz08, GKJa08]. Thus, the CoP needs to be evaluated continuously to predict future challenges and opportunities. Nonetheless, such evaluations are often time-consuming. Automating the evaluation process and providing an intuitive interface helps Communities of Practice (CoP) implement continuous success evaluations into existing CIS, which in return helps them find ways to improve the community [Renz08].

Success modeling is the process of formalizing the success of a CIS [Renz16]. Current success modeling techniques are complicated web interfaces, which require technical know-how to use them. On the other hand, success modeling is a social process in which all community members should participate. However, current web interfaces are often not optimized for collaboration. Furthermore, the use of smartphones has not been taken into consideration by classic web interfaces. Thus, new interfaces are required, which cope with the form factor of hand-held devices. Those interfaces need to provide simple and intuitive collaboration possibilities to include all community members in the success evaluation process.

1.2 Thesis Goals

Social networks are a popular tool for information exchange and social interaction inside a community. Using a chatbot inside a CIS as an interface is easier to learn and seems more natural to non-experienced users. Chat platforms are optimized for smartphones and other hand-held devices. Therefore, the use of chatbots might address the preceding issues.

Accordingly, we created a social bot to define and visualize *success models* for existing CIS. The social bot provides an interface to communicate with the Mobile Community Information System Oracle for Success (MobSOS) system.

The bot was modeled with the Social Bot Framework and deployed with the las2peer Social Bot Manager. Users can add the bot to a chat platform that allows bots, such as Slack or Telegram. The bot can be integrated into group conversations to allow members of a community to collaborate more easily.

The main focus is to combine the MobSOS framework with the Social Bot Framework. We want to determine if such a service improves the communities' collaboration and *success awareness*.

2 State of the Art

This chapter provides an overview of the concepts and technologies that are necessary for our current work. First, we define domain-specific terms in section 2.1. We describe the technologies used in the implementation in section 2.2.

2.1 Background

This section introduces the concepts, which are relevant to the implementation part of this thesis.

2.1.1 Social Bots and Chatbots

Bots are interfaces, providing automated services to end-users. In contrast to traditional computer programs, bots can use the same services and perform the same actions as human users. Those actions include browsing web pages or issuing API calls.

> “A social bot is a computer algorithm that automatically produces content and interacts with humans on social media, [...]” [FVD*16b]

Social bots are interacting with users in human-like conversations. Social bots are getting more and more popular in Human-Computer Interaction (HCI), as they integrate automation into the daily lives of people [BFPN17]. Users interact with a social bot through a dedicated channel like the chat functionality of a social network. Social bots, which use chat platforms, are called chatbots, or conversational bots [WWX*16, AAA17]. Chatbots have recently become very popular in the context of customer care [CHW*17, FVD*16b], where they act as virtual assistants, which can answer simple questions [CaWh14] or perform predefined tasks. Their simplicity allows even non-experienced users to perform actions of various degrees of difficulty. Chatbots can assist companies in customer service, allowing staff to concentrate on less mundane issues with customers. This increases the effectiveness of the service while lowering costs [AAA17].

We can distinguish between retrieval-based chatbots and generic chatbots [NLKl19, WWX*16]. Retrieval-based bots are more focused on getting specific tasks done than engaging in a coherent conversation. Retrieval-based chatbots measure the similarity of user queries to candidate responses and run the task that is defined for that response.

Generic chatbots, on the other hand, also produce generic responses. An advantage of generic chatbots is that they provide a better user experience as users can interact more naturally with them than with retrieval-based chatbots.

Chatbots use natural language processing techniques to determine what the user wants to do. Chatbots provide a better user experience [CHW*17] than traditional command-based tools because the user is not required to memorize commands to get the desired results. Thus, even non-trained users can use such chatbots. Users can interact with them in a familiar environment. Therefore, chatbots are desirable as assistants.

Natural Language Understanding

Natural Language Understanding (NLU) is a branch of Artificial Intelligence (AI), which aims to make computer programs understand human language. NLU transforms a block of input text into a data structure that is programmer-friendly but still describes the original meaning of the text [CWB*11].

Current NLU algorithms look only at the syntax of words, which relies on statistical features, such as word occurrence frequency [CaWh14]. However, humans consider far more information. The reading of a word triggers related concepts and experiences. Those experiences are combined to deduce the meaning of a text.

Computers try to close this knowledge gap with computational models, which emulate human cognitive processing. By evaluating the results of those models for a given input, researchers can incrementally improve the accuracy of the model [CaWh14]. NLU is a domain that is a key research topic for companies like Google and Facebook [AAA17].

2.1.2 Communities of Practice

A Community of Practice (CoP) is a special kind of community consisting of members studying a particular domain.

> “ Communities of practice are groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly.” [Weng98]

In contrast to traditional knowledge systems, where a student passively receives his information, the learner is actively participating in the community by voluntarily sharing his knowledge with other members of the community [AMMi15, Kern08]. The advantage of CoPs is that they make the learning process easier [SaAr05] and the transfer of knowledge faster [CuZe05].

Members of a CoP have different degrees of knowledge of the domain. For example, a student has less knowledge than a researcher in a specific domain. Furthermore, members have various degrees of interest in the practice of the community. Members with a lot of interest in that domain tend to know more about it and tend to contribute more to the community. Such members are called *core members* of the community. Members, which are less active in the community are called *lurkers*.

Members of CoPs do not have to belong to the same organization but participate in common work through social interactions [Weng98]. In the case of an online CoP, this could be an online social network [CuZe05]. Multiple, overlapping, CoPs can communities can exist inside a single CoP.

2.1.3 Measurement of Success

Communities need to constantly be aware of their successes and failures to improve their work and adjust to current requirements. A general model for success does not exist, as communities are very diverse, dynamic, and informal structures with permeable boundaries. The members of the community thus can be active participators in the community success modeling, as the success factors are relevant to the practice of the community members [RKJa15].

The community must continuously adjust the success factors of the success model to satisfy current requirements. Success factors can be internal aspects, like project duration, cost, and quality. Success factors can also be external aspects, like customer satisfaction [AgRa06].

It is important to note that a project can be evaluated from different perspectives [RKJa15], like the developers, the customers, or the end-users. In each of these perspectives, each dimension of success can be more or less valuable. DeLone and McLean [DeMc92] base their model on six different interdependent measures:

- **System Quality** are factors that describe the technical specifications of the system, such as service response time and scalability.
- **Information Quality** are factors about content quality, such as understandability and conciseness.
- **Use** are factors about how a system is used, such as the regularity of use by individual users or for the whole community.
- **User Satisfaction** includes factors describing users' personal experiences and is thereby largely subjective.
- **Individual Impact** are factors about how a user adapts to the system over time, such as task performance.
- **Organizational (or Community) Impact** includes factors about how the system influences the organization (or community) over time.

In order to collect data for the metrics, one should follow the principle [RKJa15]:

> "observe, where possible; only survey, where inevitable."

2.1.4 MobSOS

The Mobile Community Information System Oracle for Success (MobSOS) framework is a tool, which assists online communities in monitoring and evaluating services. The MobSOS success model extends the classical IS success model of DeLone and McLean

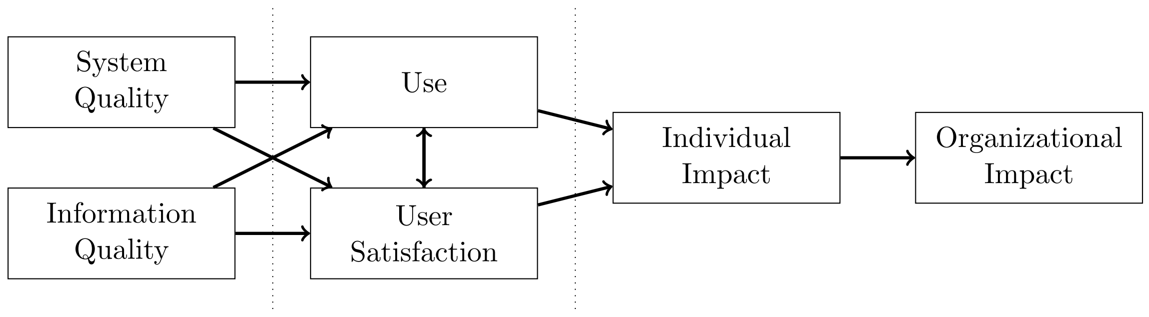


Figure 2.1: CIS Success Model by DeLone and McLean [DeMc92]

(D&M) by adding features, which were not covered by the original model, such as Quality of Service and mobility [Renz08]. The MobSOS Success Modeling concept is described as an iterative process.

The MobSOS Continuous Community Analytics (CCA) services extend the MobSOS framework by providing visualizations of MobSOS data and the ability to add Mediabases [Kers20], which can be used for visualizations.

2.1.5 Mediabase

Storing Web 2.0 data is a challenging task due to the variety of different non-interoperable formats. Mediabase is a concept that has been proposed to store Web 2.0 data in a dynamic way by delivering an analysis environment for Web Science data sets [KlPe08]. Mediabase uses graphs to model the generated data. The nodes in those graphs are called *Actors* and represent either a *Medium*, an *Artifact*, a *Member* or a *Network*. A *Medium* can be a blog or a wiki. The *Artifact* would be an individual entry. The *Network* represents the community in which the different *Members* interact. *Members* do not have to be human but are assigned a specific role in the *Network*. Furthermore, Mediabase includes *Services*, like a *Watcher* to inspect the data. A Mediabase consists of a backend database, like DB2, or MySQL, crawler scripts, monitoring processes, and a frontend for visualization of data and user interaction.

This approach is favorable for the management of large scale-free dynamic social graphs.

2.2 Systems

This section describes existing systems which will be used for the realization of this project.

2.2.1 las2peer

las2peer[1]is a platform, which allows users to deploy their HTTP services with ease. It is a distributed platform based on peer-to-peer networks without a central authority

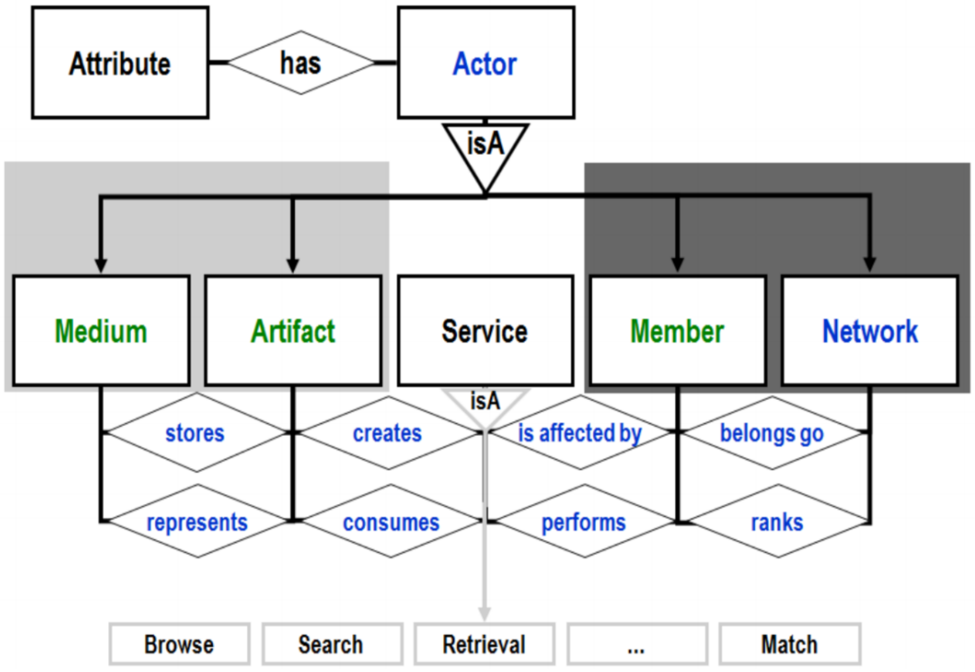


Figure 2.2: Mediabase Overview [Klam10e]

[KRdJ16]. Each service is a *node* in the network. Many nodes can form a peer-to-peer seed network, communicating through standard protocols. Data storage and communications are encrypted, adding an extra layer of security to las2peer services. This allows even small communities to design and deploy secure las2peer services.

New nodes can join the las2peer network by using the *bootstrap* flag followed by the IP address of one of the nodes in the network. The service can be addressed by its *ServiceAgent id* or by listening to an arbitrary topic. It can address other services in the network in the same way.

The services inside a network share a key-value store as a distributed hash table (DHT). The entries are stored on each node's host filesystem. The entries are encrypted by their author. Only the author can change the value of an entry.

Groups are a feature in las2peer for security and defining rights in a community. A user can create a group by generating a *GroupAgent*.

In order to communicate with the nodes of the las2peer network from outside, a *WebConnector* can be started on a node. This connector handles REST requests to the services. A service can then be accessed over HTTP by the following URL scheme:

```
http(s)://ip:port/serviceAlias/methodName
```

The *serviceAlias* is a name, which identifies the service in the network.

las2peer includes an editor for Modeling and Code Generation called Community Application Editor (CAE). The CAE follows a model-driven approach as the las2peer

[1] https://las2peer.org/

services can be created by designing a model using the *Canvas* widget. The editor can be used as a collaborative, near-real-time framework [dND*16].

las2peer also includes a framework for community evaluation based on MobSOS. This framework can be used to monitor and evaluate the success data of community applications.

MobSOS Data Processing

The MobSOS Data Processing service collects logs from the Mensa service and the MobSOS Success Modeling service. Logging data in MobSOS can be done by using the listing 2.1

```
Context.get().monitorEvent(
    MonitoringEvent.SERVICE_CUSTOM_MESSAGE_X, data
);
```

Listing 2.1: Example use of a MonitoringEvent

. The MonitoringEvent `enum` provides different events which are logged in the network. An overview of the different monitoring events can be found on GitHub [2]. Each service can use its own events by using the `MonitoringEvent.CUSTOM_MESSAGE_X`, where X is a number between 1 and 99. Those `data` variables contains generic JSON data. This data provides information about how community members are using the service. To monitor messages in the service, the service needs to be started using the `-observer` flag. The data is stored in the MobSOS MySQL database under the `MESSAGE` table. The table contains a column `REMARKS` which is used to store the JSON data.

The MobSOS Data Processing service[3]processes data, which are monitored in the context of a service in the network, and stores it in a database.

MobSOS Success Modeling

The MobSOS Success Modeling service[4]can be used in order to create and manage success models that follow the six success dimensions proposed by DeLone and McLean [DeMc92]. Each group in a las2peer network can create multiple success models for each las2peer service in the network. The groups reflect the las2peer *GroupAgent* and the services reflect the *ServiceAgents* in the las2peer network. The success model contains the six MobSOS dimensions.

```
<SuccessModel name="Name␣of␣success␣model"
    service="your.service.name@version">
    <dimension name="name␣of␣dimension␣1">
        <factor name="Name␣of␣factor">
            <measure name="No␣Factors␣or␣Measures␣yet"/>
        </factor>
```

[2] `https://github.com/rwth-acis/mobsos-data-processing/wiki/Built-In-Monitoring-Events`
[3] `https://github.com/rwth-acis/mobsos-data-processing`
[4] `https://github.com/rwth-acis/mobsos-success-modeling`

```
    </dimension>
     ...
    <dimension name="name␣of␣dimension␣n">
        <factor name="Name␣of␣factor">
            <measure name="No␣Factors␣or␣Measures␣yet"/>
        </factor>
    </dimension>
</SuccessModel>
```

Listing 2.2: Example of a success model

Each of these dimensions includes a list of success factors describing the dimension. Each factor contains a list of success measures. The success measures are stored in a separate file called *Measurement Catalog*. A measure defines how to get data and how to visualize it.

Three types of visualization are supported

- Value: This visualization returns a single value with an optional unit
- Chart: This visualization returns a chart as a picture. Different chart types from the Google Charts API[5]are available.
- KPI: This visualization uses an arbitrary amount of queries, with each query returning a value. The values from the query can be combined through various mathematical operations.

2.2.2 Social Bot Framework

The Social Bot Framework (SBF) is a Web-based model-driven near-real-time environment to develop social bots [NLKl19, Neum18]. The model-driven development allows less experienced users to take part in the development process. This type of development is favorable for creating a chatbot inside a Community of Practice (CoP). The framework consists of a GUI and a Social Bot Manager Service. The GUI of the Social Bot Framework is used to model the bot and define the actions, which the bot should perform. Recent changes of the SBF enabled bidirectional communication between the Social Bot Manager Service and chat platforms [Wies19]. The modeling space of the SBF can be used collaboratively. The modeling canvas is used to model the bot. Icons, representing elements of the bot, can be selected from the palette widget and connected with each other.

The GUI also includes an NLU Model Training helper, which can be used to write training data for intent recognition using a Rasa server. The resulting model is sent to the Social Bot Manager which deploys the bot and handles the communication with chat platforms, like *Slack* [6] or *rocket.chat*[7]. The Social Bot Manager service[8]is a

[5]`https://developers.google.com/chart`
[6]`https://slack.com/`
[7]`https://rocket.chat/`

las2peer service and can thereby call other las2peer services in the network. Thereby the bot can perform service requests to any service in a las2peer network.

2.2.3 RASA

Rasa is an open-sourced framework for building chat- and voice-based bots. The goal of RASA is to bring current advances in machine-learning to end-users, such that they can create chatbots in a simple way [BFPN17]. RASA is split into a component for natural-language understanding (RASA NLU), and a dialogue management component (RASA Core).

RASA NLU is used to extract the *intents* and intent *entities* of a user message. The *intent*[9]of a message represents the thing that the user tries to accomplish. Entities are keywords that are relevant to the current context. Intent recognition is done by using a text classification based on the fastText approach [BFPN17]. fastText has similar accuracy to conventional deep learning classifiers but is much faster and more easily scalable [JGBM16].

RASA Core consists of a *Tracker*, a *Policy* and a set of *Actions* [BFPN17]. The Tracker manages the current state of the conversation. It gets an intent as input and forwards it the Policy, which selects the next Action, given the current state of the conversation and a predefined list of actions for each intent. The Action triggers the Tracker to recalculate the new state and might also send a reply to the user.

The system is built in a modularized way, with each service providing HTTP APIs [BFPN17]. Thereby, the NLU component can be used independently of the rest of the framework, making it easy to integrate it into an existing system [RaKe19].

2.2.4 Google Charts API

Google Charts API[10]is an API that provides visualizations for database data. The data, which should be visualized needs to be wrapped inside a JavaScript class called `DataTable`, which represents a two-dimensional table with rows and columns, where each column has a datatype. A new column can be added using the `DataTable.addColumn` function, which takes the type and name as input. An example can be seen in Listing 2.3 from the Google Charts API documentation[10].

```
var data = new google.visualization.DataTable();
data.addColumn('string', 'Topping');
data.addColumn('number', 'Slices');
data.addRows([
        ['Mushrooms', 3],
        ['Onions', 1],
        ['Olives', 1],
        ['Zucchini', 1],
        ['Pepperoni', 2]
```

[8]https://github.com/rwth-acis/las2peer-social-bot-manager-service
[9]https://rasa.com/docs/rasa/glossary#intent

```
]);
```

Listing 2.3: Example use of the DataTable class

The visualizations are rendered as SVGs in the browser or downloaded as a PNG file by making a `http` call with the `getImageURI` as image URI parameter.

2.2.5 GraphQL

GraphQL[11]is a query language specification for web APIs. Instead of calling multiple endpoints for different requests, users can get all data from one endpoint by specifying what they need in a query. This prevents over- and under-fetching of data[12], thus increasing overall performance, thus, leading to a better user experience [KKK20].

GraphQL does not mandate the use of a specific programming language, the actual implementation is up to the developer of the server. [13] Many different implementations have been written for programming languages like Java [14] and JavaScript. [15]

Transforming an existing REST API into a GraphQL API can be a cumbersome task [WCL18]. That is why automation processes in the form of Wrappers have been proposed [KKK20], which convert an existing schema such as a Swagger schema into a GraphQL schema[16] [17] [18]. Such wrappers transform GraphQL requests into RESTful requests as depicted in Figure 2.3.

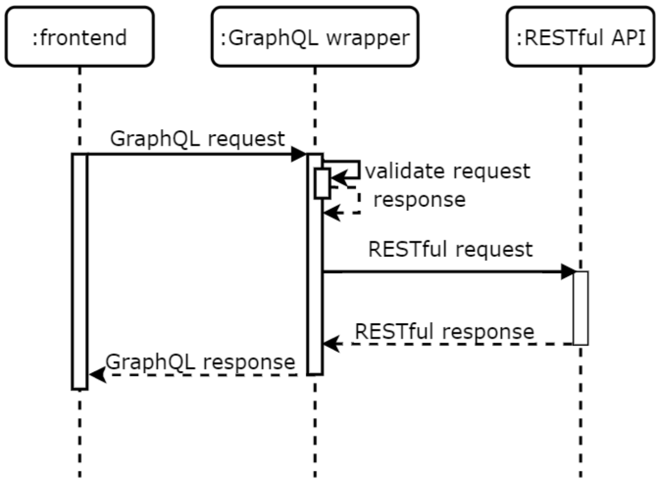


Figure 2.3: Example of a GraphQL request

[10]`https://developers.google.com/chart`
[11]GraphQL: `https://graphql.github.io/`
[12]`https://www.howtographql.com/basics/1-graphql-is-the-better-rest`
[13]`http://spec.graphql.org/June2018/`
[14]Java GraphQL Library:`https://www.graphql-java.com/documentation/master/`
[15]JavaScript GraphQL Library:`https://www.npmjs.com/package/express-graphql`
[16]`https://github.com/yarax/swagger-to-graphql`
[17]`https://github.com/IBM/openapi-to-graphql`
[18]`https://github.com/rwth-acis/openapi-link-generator`

In GraphQL, data objects are defined as a *type*. GraphQL provides simple base-types like Int Float, Boolean, and String, but new types can also be defined. An example can be seen in Listing 2.4.

```
 type REVIEW {
   dishId: String
   mensaId: String
   timestamp: String
   stars: Int
   comment:String
   author: String
   id: ID
 }
```

Listing 2.4: Example of a GraphQL schema

New fields can be dynamically added without influencing existing queries. The schemas are formalized as either a *Query* or a *Mutation*. They are declared by the keyword *schema*. The query keyword can be omitted. An example of a query can be seen in Listing 2.5.

```
{
 REVIEW(id: "1000") {
   stars,
   author
 }
}
```

Listing 2.5: Example of a GraphQL Query

Here we access the rating of the review with the ID 1000.

Mutations are used to modify data. An example of a query can be seen in listing 2.6.

```
mutation addReview(){
 {
 author
       stars
 }
}
```

Listing 2.6: Example of a GraphQL Mutation

2.2.6 Docker and Kubernetes

Docker[19]is an open-source containerizing technology, which aims to simplify and accelerate the development of web services. A Docker *container* is a standalone executable

[19]`https://www.docker.com/`

package that contains all dependencies for the actual application. Docker containers run using the Docker *engine*. Docker *images* define the way a container should run, while the container is the actual runtime. In contrast to virtual machines (VMs), Docker containers are virtualizations of the application layer instead of the physical hardware layer. Thereby, they take up less space (only a few MBs) and are faster to boot. Docker images can be uploaded to Docker registries[20], which are central repositories for Docker images.

Kubernetes[21](K8s) is an open-source container orchestration tool, which automates deployment and scaling of a containerized application, like Docker containers. K8s introduces an abstraction layer to containers called *pods*. Pods can contain multiple containers. Pods are created and managed automatically by K8s. K8s also introduces *services*, which serve as the communication endpoint of the pod, as it maps a single IP address to a pod. This is necessary as the IP address of the pod can change if it is restarted. The different applications can communicate with each other using `http`. Different services and pods can be grouped together in *name-spaces*. K8s also has a service for DNS resolution. Each service gets its own domain in the form `my-service.my-namespace`.

[20]`https://docs.docker.com/docker-hub/`
[21]`https://kubernetes.io/`

3 Concept

3.1 Scenario: Mensa Community

We illustrate the software system by considering a community of frequent canteen visitors. We will call this community the Mensa Community. The success principle can be applied to this community because community members share a common interest that they are passionate about: the food at their local canteen [Kers20]. The success factors for this community describe how users are interacting with the service. The members of the Mensa community are students and university employees.

The community has a hierarchical structure consisting of different community levels. In our case, the top-level represents the community of all canteens in Germany. The intermediate level represents the local Mensa community of a particular university. The lowest level of the community represents a circle of friends, which frequent the canteen together. Note that members can belong to multiple circles of friends. Thus communities can be overlapping.

Members of the community use community applications [1] [2] to access information about the Mensa and their community respectively.

Members of the community can interact with a chatbot called *mensabot*. The mensabot is modeled using the Social Bot Framework. Users interact with the bot via chat in the Slack workspace of the community. They can ask the bot about the food served on a given day.

As members of the Mensa community are very passionate about the food, they can also use the bot to add a review of their meal.

Furthermore, members can query visualizations about the success of the community through the chat application. Those visualizations include current success metrics defined by members of the community. The metrics are defined using the MobSOS Evaluation Center [Hoss19]. Members can add measures to the success model of the community with the help of the bot.

3.1.1 Use Cases

Get the menu for the canteen

Alice wants to decide whether to go to the canteen on a given day. She opens the Slack app and asks the bot for the menu at the canteen on that day. The bot looks up the menu for that canteen. The resulting menu is displayed inside the chat. Each

[1] `https://tech4comp.dbis.rwth-aachen.de/mensa/`
[2] `https://monitor.tech4comp.dbis.rwth-aachen.de/`

dish includes an average rating of reviews made by the community. The reviews help Alice decide on a meal.

3.1.2 Setting a default city

Bob asks the bot about the menu for a canteen called *Mensa Academica*. The bot finds two canteens matching that name. One is located in Aachen, while the other one is located in Leipzig. The bot asks Bob to clarify which canteen he meant by displaying both canteens in a list. Bob chooses the second item in the list. The bot returns the resulting menu. The bot asks him if he wants to save the city, in which his canteen is based as his default city. This helps the bot to identify the canteen on further requests better.

Adding a review

Erdzan just finished his meal at the canteen. To his surprise, the food was better than expected. He decides to do a positive review using the mensabot. The bot will start a dialogue by asking questions about the meal. Each question results in a data point for the review. The bot saves the review and displays the result in the chat.

The bot asks him if he wants to upload a photo of his meal. He chooses to do so. The final review is then added to the database and included in future evaluation requests.

Issuing a visualization request

Aaron is unsure of whether to go to the canteen. He asks the bot how popular a specific canteen is. The community had previously defined *popularity* as a measure in the success model. The measure records the number of requests for that canteen's menu and ranks it according to the other canteens. The bot recognizes this as a predefined measure contained in the success model. It looks up information about the measure in the measure catalog.

The measure contains information on how to get the data and how to visualize it. This includes information about the database and the corresponding SQL query on that database. The bot transforms this information into a GraphQL request and sends it to the CCA system's GraphQL API. The GraphQL server returns the appropriate data.

The bot then visualizes this data and displays the resulting visualization inside the chat. After the user has seen how good the food is, they are asked by the bot whether they will go to the canteen. They tell the bot that they will go.

Success Awareness

Students in the Mensa community are aware that their community's success depends on the active participation of its members. They are discussing the success of the community inside a group chat. They want to know how active the members of the community are. One student calls the bot, which is a member of the chat group, by name. They tell it to visualize the average time that a user is interacting with the service. The bot recognizes this as a measure and runs a predefined query. The resulting data is visualized in the group chat.

Success modeling

The students see a trend in the visualization, which indicates that fewer people are using the bot. The students blame the Coronavirus for this negative trend as the canteen can only be used as takeout. Furthermore, many students returned to their hometowns. The students want to verify that the lack of activities is due to the Coronavirus and not decreasing food quality. They ask the bot to update the success model of their community. They add a measure that models the average stars of reviews over time. After visualizing the new metric, they observe that the average stars have remained more or less constant over time.

The students are asked to add a comment to the change so that community members can understand the reason for the change at a later date.

Modifying a review

Alexander just added a review to the database. He realizes that he forgot to mention the excellent quality of the fries. He asks the bot if he can still update his review. The bot will ask the user the same questions and, afterward, update the entry in the new review database.

Evaluating the service

Peter has been using the bot for some time now. The bot asks questions about the quality of the service. Such questions include how satisfied he is with different aspects of the service or how likely it is to recommend it to a colleague. The results are used for measures in the User Satisfaction dimension.

Adding a new database

Core members and moderators of the community can add a new information source by adding a new database to the system. They do it using the frontend of the CCA service and the MobSOS query visualization service. The moderators are aware that many members of the community also add reviews on sites like Google reviews. They add a Mediabase containing reviews collected by crawlers from sites like Google Reviews. Those reviews can now also be used for success modeling.

Visualizations of multiple databases

Multiple databases can exist in a system. Apart from the reviews added by chat, other reviews are also collected with crawlers. A student requests a visualization of a

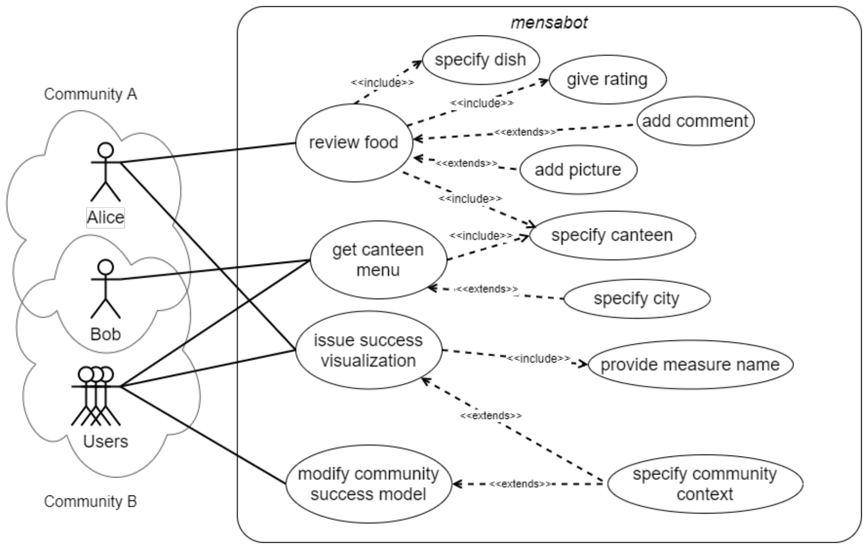


Figure 3.1: Use case diagram for the Chatbot

metric that represents the overall success of the community. The data is aggregated from different sources, like reviews on Facebook, Google, stored in the Mediabase, and reviews made inside the las2peer system. The resulting visualization contains all databases' results to define how popular the specific canteen is.

3.2 Requirements

The requirements are derived from the use case. We distinguish between functional and non-functional requirements.

3.2.1 Functional Requirements

Retrieving Data from GraphQL API

The system needs to be able to make queries to the GraphQL API, to get data, which will be used to visualize success metrics. The GraphQL API for the Continuous Community Analytics systems provides data from different data sources, such as a las2peer database, a Mediabase with data collected from outside the las2peer system, and a database contains service logs.

Get Menu for a canteen

The bot needs to be able to get the menu for a specific canteen for the given day and display it inside the chat as a list of text items. The service should be able to use the OpenMensa API [3]to get the menu for canteens all over Germany.

[3]`https://doc.openmensa.org/api/v2/`

Enter Review Context

The bot should be able to recognize if a user wants to start a review. This can be done with the help of intent recognition from a RASA server. Adding reviews should only be possible in private chat with the bot.

Group Chats

Users should be able to add a bot to a group chat. Services, called by the bot, should be able to distinguish between private- and group-chats.

Restrict certain actions

Certain actions should be restricted based on the user that is requesting the action. Updating the success model should only be possible by a member of the community.

Listen to Mentions

Inside a group chat, the bot should only respond to messages which specifically mentioned the bot. This is important to prevent the bot from spamming public channels.

Use Google Charts API

The data retrieved from the GraphQL API needs to be passed to the Google Charts API to create a visualization. The resulting visualization should be a picture so that it can be represented inside a chat. The visualization can be rendered by using an HTML image generator.

Cope with spelling mistakes

The bot should be able to understand the user, even if they made a spelling mistake.

3.2.2 Non-Functional Requirements

Usability

The bot should be easy to use. Non-trained users should be able to interact with the bot with ease.

UI optimization

The bot should be designed for chats which will mainly run on smartphones. As such, visualizations should be easy to read, even on small devices.

Compatibility

The bot should be extensible to any chat platform, which allows the use of bots.

3.3 System overview

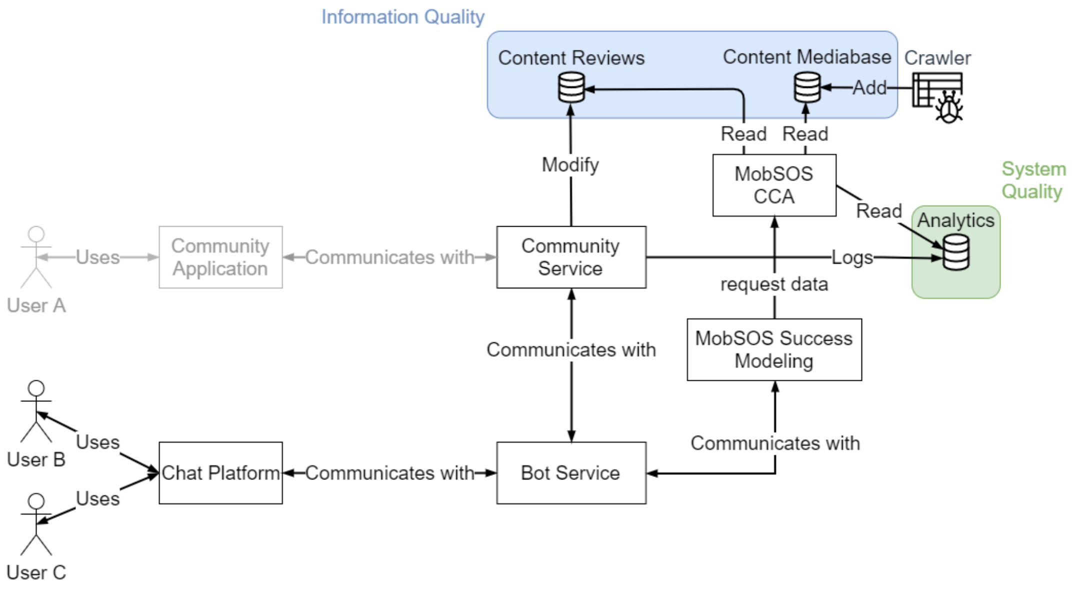


Figure 3.2: System overview

Figure 3.2 shows how the *Bot Service* can be integrated into an existing community information system. Users communicate with the *Chat Platform*, which transmits requests to the Bot Service.

The Bot Service communicates with the *Community Service* to provide users the same, or similar, information as an existing *Community Application*. The bot service also communicates with the *MobSOS CCA* system to retrieve success visualizations and display them as charts on the Chat Platform. Users can also use the chat to model success queries and modify the existing success model.

Both the community service and the bot service generate logs from requests. The logs are stored in a database representing the *System Quality* dimension of the MobSOS success model. The MobSOS CCA system can retrieve those logs. Furthermore, the MobSOS CCA system also has access to various content databases that represent the *Information Quality* dimension of the MobSOS success model.

4 Realization

In the following chapter, we describe the realization of the software system. In section 4.1, we give an overview of the core components and how they interact with each other. In section 4.3, we describe additional components. Figure 4.1 shows an overview of the core components. Additional components from section 4.3 are omitted to prevent cluttering.

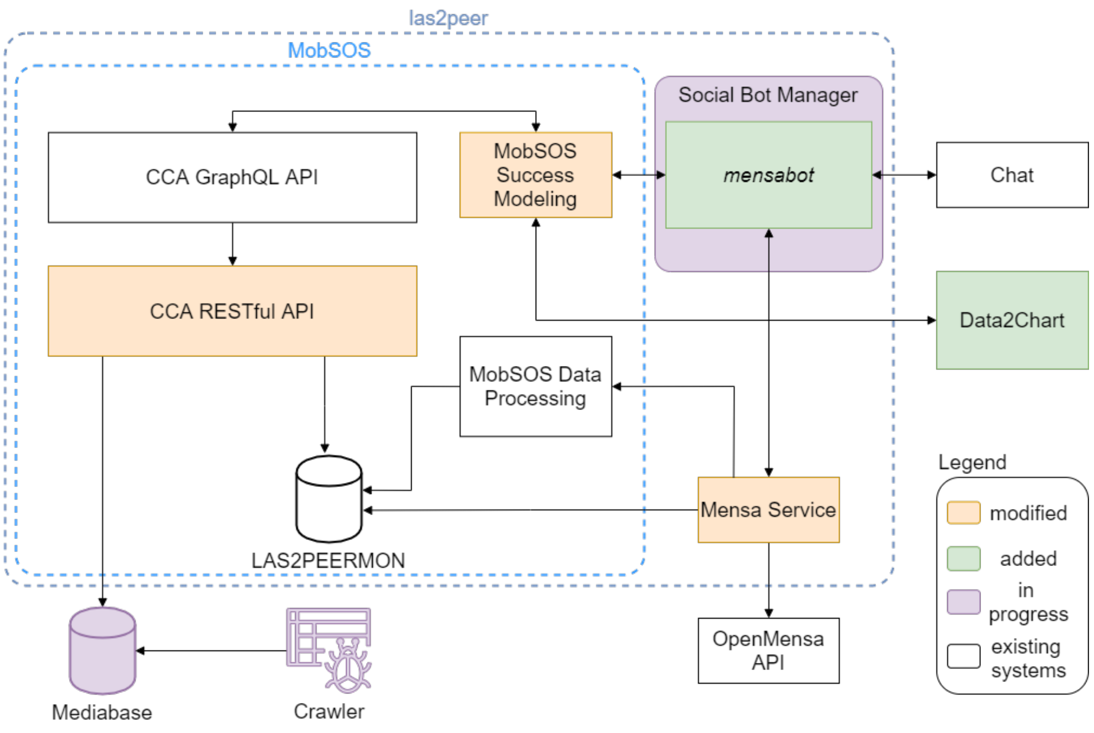


Figure 4.1: Overview of the different components

4.1 Structure

The system consists of three major components: the *Social Framework*, the *Mensa Service*[1] and the *MobSOS* services.

[1]`https://github.com/rwth-acis/las2peer-Mensa-Service`

4.1.1 Social Bot Framework

The Social Bot Framework is used to model and deploy the *mensabot.* The mensabot is created in the space *mensabot* of the Social Bot Framework.

We created the bot model using the canvas of the frontend. The frontend also serves to write the intents to train an NLU model for intent recognition. The bot should be able to be added to group channels. The bot should act as a quiet agent listening to mentions and filter out the rest of the conversations. A mention has the following form `@botname`, where *botname* represents the username that the bot has in the Slack channel.

4.1.2 MobSOS services

The MobSOS services include the MobSOS Data Processing, MobSOS Success Modeling, MobSOS CCA GraphQL, MobSOS CCA REST services.

MobSOS CCA GraphQL and REST

The MobSOS CCA backend service has access to three different databases. First, the MobSOS CCA service can access the food reviews stored inside the las2peer database of MobSOS.

Second, the system can access *Mediabase* data, which contains data from reviews made outside of the las2peer system. This data could be collected by a crawler bot, which searches Google and Twitter for reviews of the canteen. Those two databases provide mostly data for the Information Quality factors of the MobSOS success model.

Third, the service can also access monitoring data generated by the Bot and the Mensa Service, stored in the MobSOS database.

Initially, the REST service fetched data from the database by providing a SQL query and the database and schema on which it should run. It would then create a connection to that database, execute the query and create a JSON object from the `ResultSet`.

The problem with this approach is that the order of the attributes specified in the `SELECT` statement of the query is lost. However, this order is crucial when making visualizations. We considered two possible solutions:

1. Modify the service to provide the data directly as an array of arrays.
2. Send order information along with the response and then use it to sort the data.

In order to still comply with the GraphQL specification, we went along with option 2.

MobSOS Success Modeling

The measure definition was modified by including additional information about which database the query is executed on. Listing 4.1 shows an example. The `data` element contains the SQL query under the `query` element and information about where to execute the query contained in the `database` element. The attributes for the `database` element are `name` and `schema`. The name identifies the database on the GraphQL server. The schema attribute describes which database schema to use. If the database attribute is omitted, the default database name and schema are used, which are `las2peer` and `LAS2PEERMON` respectively. Thereby, the community can visualize data from any database supported by the GraphQL service.

```
<measure name="Your measurement">
    <data name="Your dataset">
        <query name="countOfMessage">
            SELECT COUNT(*) as number FROM MESSAGE
        </query>
        <database name="las2peer" schema="LAS2PEERMON"/>
    </data>
    <visualization type="Value">
        <unit/>
    </visualization>
</measure>
```

Listing 4.1: example of a Measure

In the case of a chart visualization, the MobSOS Success Modeling service uses the Data2chart service. This service transforms the data into a picture of a chart in base64 encoding. This encoded picture is sent back to the success modeling service, which sends it back to the social bot manager.

4.1.3 Data2Chart

We wrote this service[2]to render google charts as an image that could be sent to the bot and visualized in the chat. We developed this service as an alternative to the MobSOS query visualization service, which did not provide the option to render the chart as an image. The service is written in NodeJS and uses an NPM library called google-charts-node[3]. The google-charts-node module creates an HTML file that loads the google charts and then renders that HTML file in a headless browser [4]and returns the resulting image. The MobSOS success modeling service calls the NodeJS server if a bot is requesting a visualization. It includes the data for the graph in a `POST` request, which has the following form:

```
POST http://localhost:3000/ HTTP/1.1
content-type: application/JSON

{
    "data": [...],
```

[2]https://github.com/lakhoune/image-renderer
[3]https://www.npmjs.com/package/google-charts-node
[4]https://www.npmjs.com/package/puppeteer

```
    "chartType":"BarChart",
    "options":{
        "title":"Chart title",
        "hAxis":{
            "title":"Axis title"
        }
    },
    "clean":true
}
```

Listing 4.2: Structure of a request

Under the data key, an array of data points. Each data point can be in the form of an array or JSON object. The chartType specifies the type of the chart. It uses the same format as the Google Charts API. Options also follow the same structure as defined by the Google Charts API. The clean attribute specifies whether the data should be cleaned. The Google Charts API expects each data point to be of the same length. Cleaning in this context means that each data point with fewer attributes than the data point with the maximum number of attributes is removed. The data will be cleaned unless this attribute is specifically set to true.

4.1.4 Mensa Service

The Mensa Service provides information about the community to the bot. This information includes canteens, dishes, and reviews. The Mensa service logs MobSOS data used to get insights into how the Mensa community uses it.

We also extended the Mensa service to use the OpenMensa API [5]. Using the OpenMensa API has one drawback. The menu is displayed only in the canteen's local language, so we cannot get the dishes name in English anymore. However, it also has many advantages:

- We are not limited to canteens in Aachen but can get canteens from all over Germany, Luxembourg, and Austria.
- We do not have to scrape the webpage of the Studierendenwerk [6], which is prone to errors if they change their frontend.

Initially, the service stored canteen-related information in the shared node storage. The service was modified to use its own database. Using a database allows us to make data persistent and make it available for MobSOS related success visualizations. Therefore some tables were added to the `LAS2PEERMON` schema:

- `Mensas`: This table contains information about canteens, supported by the OpenMensa API. It contains the columns:

[5]`https://openMensa.org/`
[6]`https://www.studierendenwerk-aachen.de/`

  - id (int PK): The id of the canteen
  - name (varchar(255)): The name of the canteen
  - address (varchar(255)): The address of the canteen
  - city (varchar(255)): The city in which the canteen is located

  The entries are updated once a month, as the list of canteens does not often change [7]

- `Dishes`: This table contains information about the dishes served at canteens. Whenever the service fetches a new menu, it checks whether it had not been fetched in the last six hours. If not, the entries are updated. The table contains the following columns:
  - id (int PK): The id of the dish.
  - mensaId (int PK): The canteen's id at which the dish was served. The id references a key in the `mensas` table
  - name (varchar(255)): The name of the dish.
  - category (varchar(255)): The category of the dish. Categories do not seem to be structured. Each region of canteen seems to have its own category names.
- `Reviews`: This table contains reviews of food made by the community. It contains the following columns:
  - id (int AI PK): The id of the review.
  - author (varchar(255)): The author of the review. For reviews made by the chatbot, the email address of the user is used.
  - mensaId (int): The id of the canteen at which the food was served. This attribute references a canteen in the `mensas` table
  - dishId (int): The id of the dish. This id references a dish in the `dishes` table.
  - timestamp (timestamp): The timestamp at which the review was recorded
  - stars (int): The overall satisfaction with the food. The values range from 1 to 5.
  - comment (varchar(255)): The user can choose to add a comment to his review.

We added the tables to the LAS2PEERMON schema, but in the future, we could set up a separate Mediabase. Ids seem to be unique for each dish at each canteen, each day it is served. To reduce overhead, we should define `name`, `mensaId` as primary keys for dishes.

[7] `OpenMensaAPIspecification`

4.2 Social Bot

The bot acts as an interface between the user and the las2peer backend services. The bot is called *Mensabot*. Chat messages from users are sent to the bot manager service. The bot manager uses a RASA server for intent recognition. The RASA server also extracts the name of the Mensa and the city in which it is located as possible intent entities from the user input.

Certain actions trigger a service call to the *Mensa Service*, or the *Success Modeling Service*. The services get access to the user message. The services also get access to context information like the intent and the entities recognized by the RASA server.

Furthermore, they also get access to meta-information from the chat platform like the `channel_id` of the Slack channel or the chat user's email address. The email address is used to store user information in the service. We added a static HashMap, which maps email addresses to context information. The context information for a particular user is a `JSON` object containing itself generic `Objects` referenced by a `string` key. Storing this information inside the service is important for a REST service, as the context is lost on the next service request.

4.2.1 Communication with the Mensa Service

The first feature, which was implemented was querying the menu for a specific canteen. The process is depicted in Figure 4.2. The user asks the bot for the menu at a specific canteen, which results in a triggered *Bot Action*. This action calls the las2peer Mensa service.

Possible intent entities recognized at this step are the *name* of the canteen and the *city* in which the canteen is located. If the canteen's name is not recognized, the bot will ask the user again to specify it.

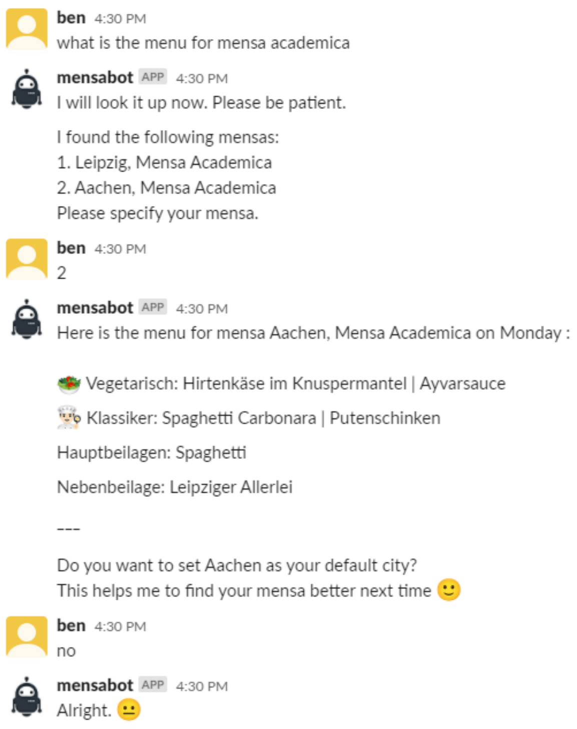
The city name can be omitted. In this case, multiple canteens may match the name provided by the user. In that case, the bot will provide a list of all matching canteen names and then ask the user to choose one of the canteens from the list. The list is saved in context so that the user can provide a number, and the service can then select the appropriate canteen from the list.

After fetching the menu, the bot asks the user if they would like to set the city where the canteen is located as their default city. If the user types yes, then this city is saved in the context of the user. On further requests, this information can help extract the right canteen if more canteen names are possible. We chose to set the default city instead of the default canteen students from one Mensa community might visit different canteens in the same city and less likely to change cities.

Next, we implemented the review process as a question-based dialogue. An example can be seen in figure 4.3.

The bot starts by asking the user which canteen the user went to and what category of dish they had. If the user did not specify a canteen but had previously asked for the menu, then the Mensa service gets the canteen from the `ContextInfo`.

The available intent entities are the name of the canteen and a dish's category (e.g.,



ben 4:30 PM
what is the menu for mensa academica

mensabot APP 4:30 PM
I will look it up now. Please be patient.

I found the following mensas:
1. Leipzig, Mensa Academica
2. Aachen, Mensa Academica
Please specify your mensa.

ben 4:30 PM
2

mensabot APP 4:30 PM
Here is the menu for mensa Aachen, Mensa Academica on Monday :

🥗 Vegetarisch: Hirtenkäse im Knuspermantel | Ayvarsauce

👨‍🍳 Klassiker: Spaghetti Carbonara | Putenschinken

Hauptbeilagen: Spaghetti

Nebenbeilage: Leipziger Allerlei

Do you want to set Aachen as your default city?
This helps me to find your mensa better next time 🙂

ben 4:30 PM
no

mensabot APP 4:30 PM
Alright. 😐

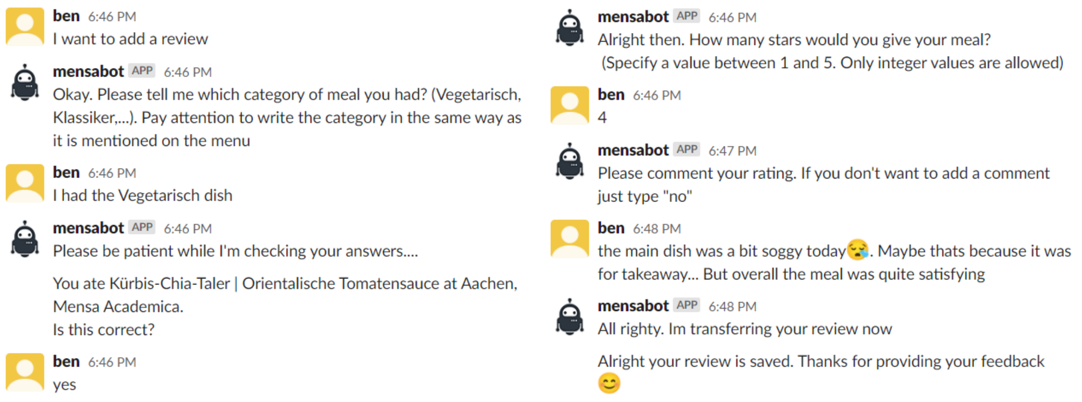
Figure 4.2: Get canteen related information

Klassiker/ Vegetarisch). The bot then extracts the dish from the menu and asks the user if the canteen and dish were correctly selected. Currently, the service only extracts dish categories by using simple string matching. Furthermore, the dish categories need to be provided in the same way they are listed on the menu. In our case, this means that they need to be provided in German.

If the user confirms the selection, then the process continues. The service stores the canteen and dish in the context. If the selection was wrong, the bot asks the user to retry, specifying the canteen and dish. The service resets the context to the previous step. In this case, this also includes removing the dish and canteen from the user context in the service.

After the user confirmed canteen and dish selection, the bot continues to ask the user how many stars (between 1 and 5) would give their meal. The bot asks the user to add a comment to the review. The user may or not add a comment. If the user types *no* into the chat, the bot recognizes this as a rejection intent and adds a review without comment.

We implemented both of these features first, as this allowed us to start collecting food reviews and service logs, which can then be evaluated later with the Success Modeling service. Making food reviews should only be allowed in private chat with the bot. The menu query should be available both in private chat and in channels. For this to work, the Social Bot Manager service needs to be updated to discern between



ben 6:46 PM
I want to add a review

mensabot APP 6:46 PM
Okay. Please tell me which category of meal you had? (Vegetarisch, Klassiker,...). Pay attention to write the category in the same way as it is mentioned on the menu

ben 6:46 PM
I had the Vegetarisch dish

mensabot APP 6:46 PM
Please be patient while I'm checking your answers....

You ate Kürbis-Chia-Taler | Orientalische Tomatensauce at Aachen, Mensa Academica.
Is this correct?

ben 6:46 PM
yes

mensabot APP 6:46 PM
Alright then. How many stars would you give your meal?
(Specify a value between 1 and 5. Only integer values are allowed)

ben 6:46 PM
4

mensabot APP 6:47 PM
Please comment your rating. If you don't want to add a comment just type "no"

ben 6:48 PM
the main dish was a bit soggy today😪. Maybe thats because it was for takeaway... But overall the meal was quite satisfying

mensabot APP 6:48 PM
All righty. Im transferring your review now

Alright your review is saved. Thanks for providing your feedback 😊

Figure 4.3: Example of the review process

private and public channels.

4.2.2 Communication with the MobSOS Success Modeling service

The next feature, which we implemented, was the visualization of success measures. Users can request them by stating that they want to visualize something. This triggers the visualization routine of the bot. Figure 4.4 shows how different components are used in a visualization.

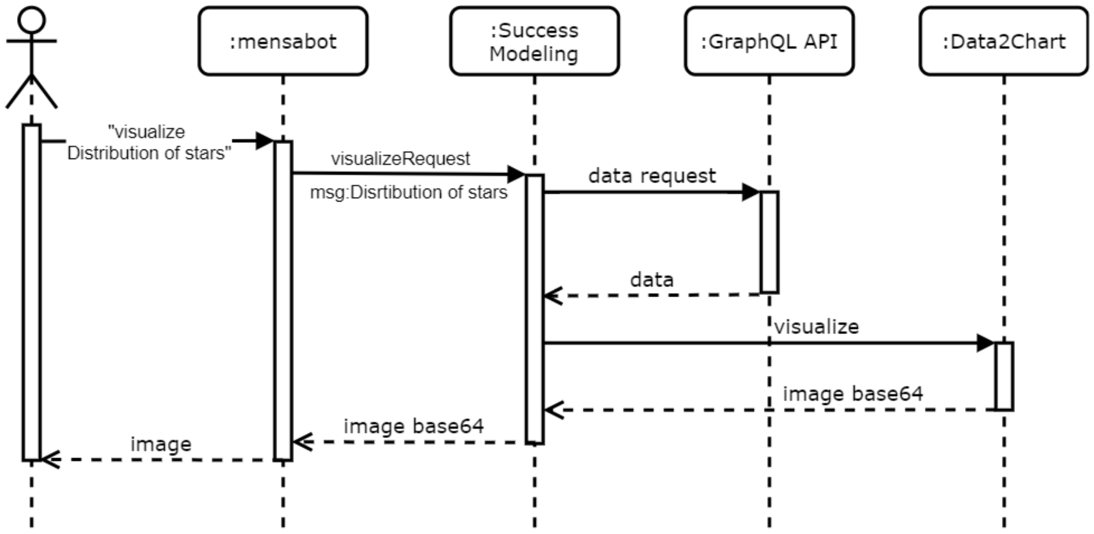
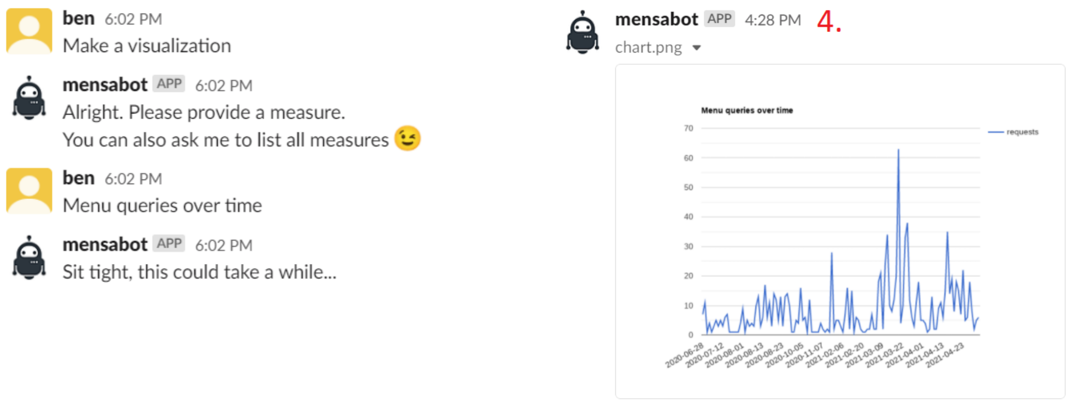


Figure 4.4: Sequence diagram of a visualization request

The bot starts by asking providing the name of a success *measure*. Alternatively, the user can also ask the bot to list all measures of the success model. If the user is asked to list measures, they can then choose one of the listed measures. The bot passes the measure name on to the Success Modeling service. The service then gets the success model for the community. This is currently done by using the default group name and default service name, but they can be specified. If a group different from the default group is chosen, the service checks if the chat user is part of the las2peer group. It is important to note here that the email address needs to be assigned to an Agent in the network. This can be done by using the las2peer frontend. Groups can also be managed under *AgentTools* on the frontend. The service looks up the measure in the measure catalog.

The service chooses the desired measure by matching the name with the name the user-provided. The resulting visualization is sent back to the bot service. If the visualization is a chart, the chart is sent as a base64 encoded image.

If no name was found, the service looks up measures with tags in the user input. The tag search results are returned to the user, and the user can choose the measure they want to visualize. An example can be seen in figure 4.5.


ben 6:02 PM
Make a visualization

mensabot APP 6:02 PM
Alright. Please provide a measure.
You can also ask me to list all measures 😉

ben 6:02 PM
Menu queries over time

mensabot APP 6:02 PM
Sit tight, this could take a while...

mensabot APP 4:28 PM 4.
chart.png

Figure 4.5: Example of a visualization request

The final feature, which we implemented, was editing the success model. The user first needs to state that he wants to update the success model of the community. Currently, elements cannot be updated. However, they can be deleted or added. To delete a factor or a measure, the user specifies it as follows

```
remove <factor/measure> [name]
```

Adding a measure is a bit more complicated. The user first selects a *dimension* which they want to add a measure to. The user then specifies the *factor* under which the measure will be appended. The user can also add a factor by providing a new name. Finally, the user chooses one of the measure catalog measures to add to the success model. The measures from the catalog are predefined using a tool like the MobSOS Evaluation Center.

We decided not to add the ability to formulate queries directly with the bot. We argue that this is not a crucial requirement because community members might not necessarily know how to write them. Furthermore, the ones that do can probably do it more efficiently by editing the measure catalog directly.

4.3 Additional services

Some additional systems were deployed or added during our work.

4.3.1 Mensa Guide

The Mensa Guide service is a frontend to the las2peer Mensa service. It provides basic functionalities like getting the menu for canteens in Aachen, doing reviews, and uploading pictures for dishes. It was updated to use the latest version of the Mensa service backend. It should also be extended in the future to use all available canteens. This should be easy to do as we have added a service function to get a list of all canteens in a particular city.

4.3.2 MobSOS evaluation Center

The MobSOS evaluation Center is a service that communicates with the MobSOS backend services. It provides a range of functionalities. Those include contact management and group management. It also allows community members to edit existing success models or create new ones. Furthermore, it also provides visualizations for the success model's measures, which it gets from the MobSOS query visualization service.

4.3.3 MobSOS query visualization service

The MobSOS query visualization service is a las2peer service that can be used to make visualizations of data in a database. It also allows users to add new databases and store SQL queries. The MobSOS Evaluation Center uses this service to get the visualizations for success measures.

5 Evaluation

In this chapter, we want to evaluate how the use of a chatbot affects a community. Specifically, we want to answer the following questions:

- Does the use of chatbots increase the success awareness of the community?
- Does it increase collaboration between members?

In order to evaluate those questions, an artificial community, as described in 3, is created. The community is using a Slack workspace called *Mensa Community* to exchange information. The members of this community are people who frequent the canteen more or less regularly. Thus, the community will consist of students, researchers, and other staff members of the university.

5.1 The requirements of the community

The first step was to determine the community's requirements and what success factors they value most. Therefore a survey was created which asked participants to order success factors based on their relevance. Furthermore, users were asked to list additional factors. A success model was then created based on the most relevant factors from the survey. Listing 5.1 shows the resulting model.

```
<SuccessModel name="mensa"
service="i5.las2peer.services.mensaService.MensaService">
    <questionnaires/>
    <dimension name="System␣Quality">
        <factor name="Low␣error␣rate"/>
        <factor name="Efficiency"/>
        <factor name="Fast␣responses"/>
    </dimension>
    <dimension name="Information␣Quality">
        <factor name="Reliability"/>
        <factor name="Accuracy"/>
        <factor name="Relevance"/>
    </dimension>
    <dimension name="Use">
        <factor name="Contributions"/>
        <factor name="Getting␣the␣menu"/>
        <factor name="Frequency␣of␣use"/>
    </dimension>
    <dimension name="User␣Satisfaction">
        <factor name="Enjoyment␣of␣use"/>
        <factor name="Number␣of␣user␣complaints"/>
    </dimension>
    <dimension name="Individual␣Impact">
        <factor name="Time␣to␣complete␣a␣task"/>
        <factor name="Time␣savings"/>
    </dimension>
    <dimension name="Community␣Impact">
    </dimension>
</SuccessModel>
```

Listing 5.1: Success Model based on requirements

A measure catalog with example measures was added for the community by us. We created the measures according to the success factors in Listing 5.1.

We planned to conduct the evaluation in two phases. In the first phase, we wanted to familiarize the participants with the chatbot. Users could have been using the bot for an extended period of approximately one month. We also wanted to evaluate the bots' performance in providing canteen-related information. We would have asked users to use the bot whenever they went to the canteen. The idea was to collect logs and reviews. We could later have used this data in visualizations of success factors. Participants would have been asked to fill out a questionnaire that covered usability questions of the service. The data from that questionnaire would be included in the success model of the community. Unfortunately, due to the Corona pandemic, the canteens were closed for most of the time. Thus, we decided not to do this first phase but to integrate it into the final evaluation.

In the final evaluation, we wanted to determine how the bot affects the community's success awareness. We wanted to find out if the bot is useful for the visualization of success metrics and if the bot encourages collaboration inside the community, especially for success modeling.

5.2 Main tasks for the evaluations

We set up the chatbot in the Slack workspace. We extended the success model of the community with some measures from the measure catalog. We asked participants to join the Slack workspace and contact the bot privately. They shared their screen to be assisted in case any issues would occur. They could ask the bot its capabilities by typing *help*. The participants performed three main tasks.

In the first task, they asked the bot to get the menu for a local canteen. This could be done with a phrase like "I want to get the menu for Mensa Academica." Afterward, they added a review for one dish on the menu. This could be done with a phrase like "I want to write a review." The bot then asked how many stars the user would give their meal and ask them to comment eventually.

In the second task, participants asked the bot to visualize different success measures from the success model. This could be done by using a phrase like "Make a visualization." They asked the bot to list all success measures and then select one from the list.

In the third task, participants asked the bot to visualize the success model. This could be done with a phrase like "Get the success model." They updated the success model. We asked them to add a measure from the measure catalog to a factor of their choice. First, the participants asked the bot to update the success model. The bot would then ask which dimension to update. The participants chose a dimension from the list. The bot would ask them which factor they wanted to add a measure. At this stage, users could either select a factor from the list or add a new one by providing a name. In the final stage, the bot provided a list of measures from the measure catalog. Participants selected a number from the list. The bot added the measure to the factor. The bot returned the resulting success model.

5.3 Evaluation results

After participants performed the main tasks, we handed out a questionnaire. The questionnaire covered questions about how participants felt about using the bot in terms of success awareness. The questionnaire included six main sections.

5.3.1 Demographics

A total of 20 participants took part in the final evaluation. Out of those, 30% had filled out the first survey about the requirements of the community. Most of the participants (90%) were between 18 and 24 years old. The remaining participants were between 25 and 34 years old. Most participants were male (85%). The participants needed to specify their **main** role 85% of the participants identified as Students, 5% identified as Researchers, and 5% as University employees. Thus all of our participants are part of the target group.

Participants were then asked, how much they agreed with a statement on a linear scale from 1 to 5. 1 meant, that they completly disagreed with the statement, while 5 meant that they totally agreed with the statement.

5.3.2 Evaluation of canteen specific tasks

This section covers the usability of the mensabot to get the menu of the canteen and write a review for a dish on the menu. It covers the aspects of the first main task. Figure 5.1 shows the results. Most participants could imagine themselves using the mensabot to get the menu and write reviews.

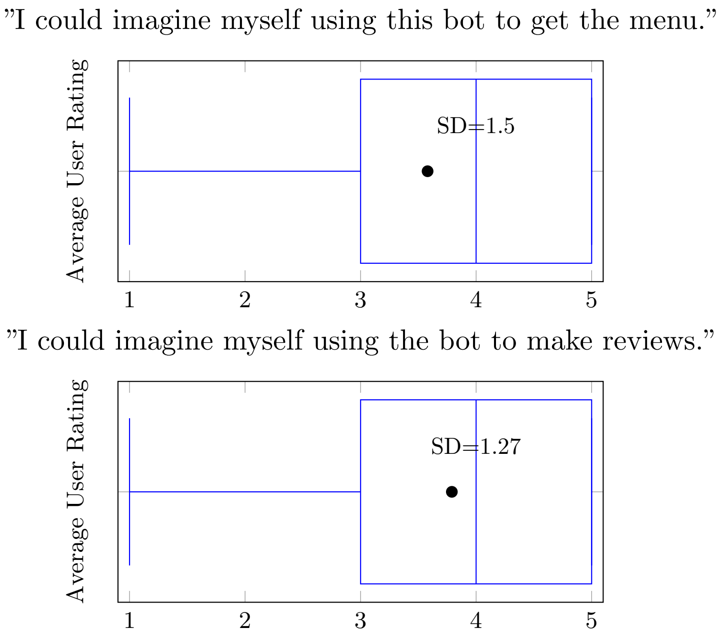


Figure 5.1: Usability of the bot to get the menu and write reviews

Some participants also noted that it would be great to see the review in the chat after submission. Some also wished to have the ability to update reviews updated at a later time.

Participants also agreed that reviews are helpful for the community, as observed in Figure 5.2. However, the awareness of the importance of reviews was higher than the desire to contribute to the community.

One participant found the procedure of adding a review a bit cumbersome, as you would have to follow a rigid number of steps, with intermediate waiting periods for the bot's answers. They suggested that users make multiple steps at once. For example, users could specify the dish, the canteen, and the stars all in one message. After that, the bot could ask for confirmation.

On some of the first evaluations, the bot did not recognize the dish category "Vegetarisch" as an entity, which meant users could not write a review for vegetarian dishes. We fixed this issue for further evaluations by adding categories like "Vegetarisch" and "Klassiker" as lookups in the training data of the NLU model. As one participant also pointed out, synonyms like "classics" should also be recognized.

We were also facing an issue with one participant. They would ask for the menu of a canteen the bot would provide a list of suggestions. The user would choose a canteen by providing a number, but the wrong canteen was selected in the service. We figured out that this issue was because the service saved the selection as a `Set` instead of an ordered list. This issue has been fixed immediately after the evaluation session.

Finally, due to the current pandemic, a lot of canteens were closed. The Mensa Service does not check yet if the canteen is open before showing the canteen in the suggestions list. Hence the bot failed to retrieve the menu, which caused some frustration. A possible fix for this would be to add a label to canteens, which are closed.

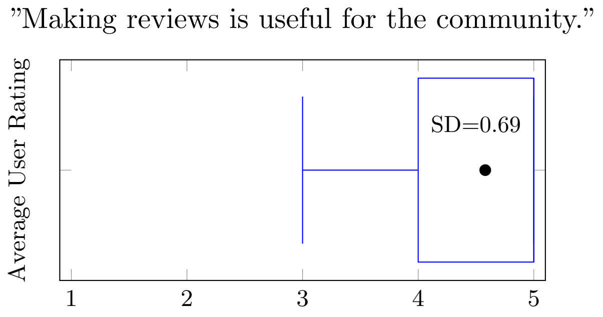


Figure 5.2: Importance of reviews to the community

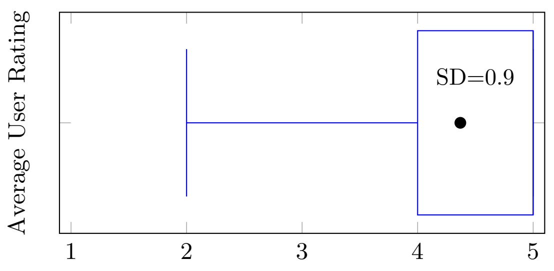
We conclude that the chatbot could have the potential to increase collaboration inside the community but does not yet fully capitalize on its potential.

5.3.3 Evaluation of visualization task

This section covers the usability of the mensabot to get visualizations of success measures. As mentioned before, we asked participants to select a measure from the success model. The resulting visualization was displayed in chat. As Figure 5.3 suggests, participants mostly understood the visualizations without needing any assistance. Some participants were overwhelmed with the number of predefined measures and found that just listing the measure name is not very helpful.

Some participants were surprised when certain measures returned a single value instead of a chart. This was because they mistook visualizations for charts.

”I understood the visualizations without needing explanations.”



”The size inside the visualizations was large enough to be read inside the chat.”

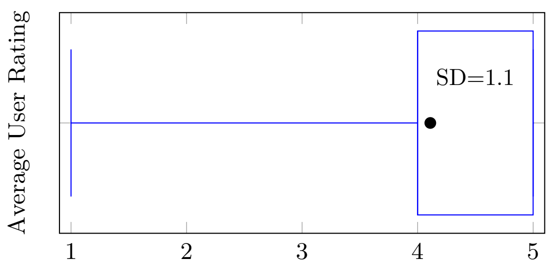


Figure 5.3: Readability of the visualizations

Overall, the visualizations seemed to be clear and intuitive to understand, even for non-technical users. Some bar-chart visualizations with long text in the y-axis resulted in the bars not being lined up correctly with the labels. Also, if there were too many labels, the axis became overcrowded, especially for line charts. Figure 5.4 shows that

”I could imagine myself using this bot to make visualizations.”

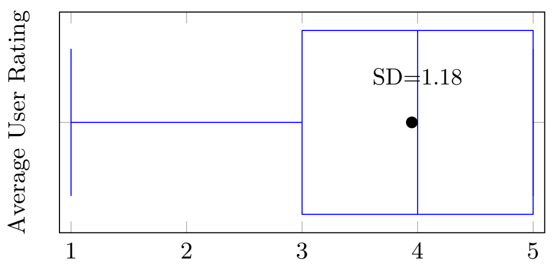


Figure 5.4: Usefulness of the visualizations

participants found making visualizations a useful feature. Considering Figure 5.5, we conclude visualizations to be appropriate to increase the success awareness of the community.

5.3.4 Evaluation of success modeling task

Figure 5.6 suggests that participants seemed to be pretty eager to add success measures to the success model.

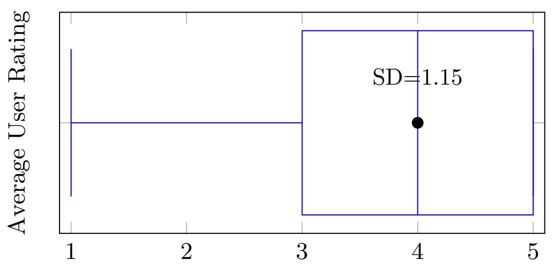


Figure 5.5: Impact of the visualizations on the success awareness

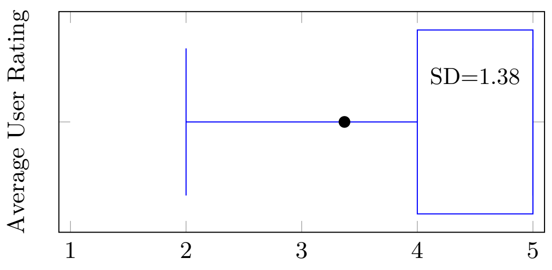


Figure 5.6: User interest in success modeling

During the evaluation, some participants were hesitant about what the different elements of the Success Model represented. They typed in the name of a factor when trying to make a visualization. One participant also had the issue that the success modeling service was stuck in the wrong context. Thus they were unfortunately not able to finish this task. Figure 5.7 suggests that participants think that success modeling is useful to the community.

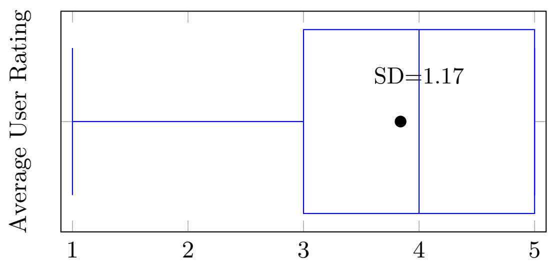


Figure 5.7: Usefulness of success modeling to the community

Overall, participants seemed to be eager to add measures to the success model. We conclude that success modeling does indeed increase the success awareness of the community.

5.3.5 General usability of the bot

For this section, we evaluated the general usability of the mensabot. We used the System Usability Scale (SUS). Figure A.1 shows the results.

Calculating the usability score[1] with the arithmetic means results in a score of 71.7, which is above the average usability score of 68 and is an Adjective Rating of **Good** [2].

5.3.6 General feedback

Participants were finally asked to leave general feedback on the chatbot. This section also includes verbal feedback during the evaluation. Participants felt that most functionalities were self-explanatory.

Participants liked that they could use natural language to communicate with the bot instead of just issuing commands. They liked that the bot could cope with small spelling mistakes. Participants also noted that the intent recognition was accurate. One participant also noted that they liked that the bot had a "personality." Another participant found the Emojis in the bots' responses satisfactory. A third participant found it helpful that they could always ask the bot for help anytime.

One participant also would have liked to communicate with the bot in German.

One participant would have preferred setting the default canteen instead of just a default city.

Participants found that the success modeling part was not intuitive. They would have preferred if the bot could have given more information on this part.

Some participants did not like the way users have to select an item for a list. Some found that there were some inconsistencies. Sometimes the bot would expect a number, and other times an item. Participants wished to have a clickable list instead or buttons where they can select their option.

The bot provided feedback if some operation might take longer or if something failed. Participants found that feedback helpful. However, one participant also wrote that "There's a general lack of feedback after commands." We assume that the participant meant the delay between the user input and the bot's first response.

5.3.7 Discussion

Through our evaluations, we have learned that most community members are aware that contributions help the community. However, they are less inclined to contribute to the community themselves without receiving anything in return. We conclude that the group of participants follows the natural structure of a CoP, which is explained in detail in section 2.1.2. Most members of the community only consume information but do not contribute to the community themselves. We learned that users are often

[1] `https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html`
[2] `https://uiuxtrend.com/measuring-system-usability-scale-sus/`

not willing to contribute to the community. We propose *Gamification* as an incentive to motivate users to contribute to the community. We have added a measure that visualizes the top contributors of the community. It could be possible to periodically display that chart to motivate users to contribute to the community. Users could receive badges when reaching defined quotas. The bot could periodically show the users' contribution to the rest of the community to motivate them to make more contributions. Another solution, brought up during the evaluation sessions, is to use nudging. The bot could ask the user to write a review if they asked for the menu earlier that day.

We found out that the chatbot can be useful to do simple tasks. Even non-experienced users can communicate with the bot, which corresponds to our definition of chatbots in section 2.1.1. Leveraging NLU, the bot can understand user intents and extract intent entities for simple sentence structures with ease. However, participants felt dissatisfied with the bot as soon as more complex tasks with multiple steps are needed. Furthermore, the more tasks we define for the bot, the slower the intent recognition will be, and the less responsive the bot will be. Furthermore, users can pronounce intents in many different ways. It is difficult to capture all possibilities. Therefore, as mentioned in section 2.1.1, NLU needs to go beyond analyzing a sentence's syntax. It should rather focus on narrative understanding by emulating how a human brain is processing natural language [CaWh14].

6 Conclusion and Future Work

6.1 Summary

In our work, we analyzed the impact of a chatbot on collaboration and success awareness in the community. We have developed a chatbot that provided community-related information and served as an interface for success modeling. We have evaluated the chatbot in an existing community.

We found that the use of a chatbot increased the success awareness of the community. Users were able to inspect the visualizations without any further help. We also found out that the bot increases contributions in the community. Contributions might further increase by other means like Gamification. We confirmed that chatbots are favorable for quick and small tasks. Nonetheless, we noted that complex tasks were less intuitive and that intent recognition is challenging for complex sentence structures.

Thus we conclude that chatbots can be a great tool to assist a community and increases the success awareness of non-experienced users.

6.2 Improvements and Future Work

As mentioned in our evaluations, the bots' usability can be improved. Users frequently need to select an item from a list of options. The list of options should be enumerated, and the user can provide a number to select the item. This has been done for some tasks and should be easy to implement for the remaining tasks. The items in the list should be clickable. The user can select an option by tapping on it. The user should also be able to type the desired item from the list directly into the chat. We could use text mining techniques to comparing the users' input to the items from the list and select the most fitting one.

The review process needs to be revised. Users could add reviews more quickly by describing which meal they had and how many stars the meal was worth in one sentence. Consequently, users could add reviews more quickly. The resulting review should be shown after it has been saved. Moreover, it should also be possible to allow users to update them in the future. Lastly, the bot could also provide reviews for certain dishes.

The mensabot could also use recommender systems to show recommendations for certain dishes. The bot could study the users' behavior for writing reviews. If the user tends to add reviews for a particular dish, the bot could alert the user if that dish is served again in the future.

We should extend the Social Bot Manager to allow bots and services to distinguish between group chats and direct chat. It should also be possible to restrict certain actions in group chats. For example, the mensabot could be added to a group chat to get the canteen menu, but only allow writing reviews in private chat. It should also be possible to filter certain messages. The bot should only respond to messages containing a keyword, e.g., the mentioning by name (*@mensabot*).

We should extend the success modeling service to generate visualizations from the success model and display them periodically in chat. The bot could periodically generate *reports*. Those reports include all the visualizations from the success model. The MobSOSb Evaluation Center needs to be updated to use the new data format for measure definitions. We could also use the Slack API to get workspace analytics[1]. Those analytics could be added to MobSOS to get more community insights.

We need to fix the picture upload for the Mensa Service. We could use the las2peer FileService to store pictures in the network. The bot can then also be extended to support picture upload. We also need to set up a Mediabase and a crawler such that reviews can be collected from outside the las2peer network. Furthermore, we could use sentiment analysis on newly added reviews to categorize reviews. This would allow for a detailed analysis of the User Satisfaction dimension of the success model.

Finally, to increase contributions in the community, we should use *Gamification.* A possibility would be to combine the las2peer Gamification framework[2].

[1]`https://api.slack.com/methods/admin.analytics.getFile`
[2]`https://github.com/rwth-acis/Gamification-Framework`

Bibliography

[AAA17] A. M. Rahman, A. A. Mamun, and A. Islam. Programming challenges of chatbot: Current and future prospective. In *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 75–78, 2017.

[AMMi15] Peyman Akhavan, Babaeianpour Marzieh, and Masoumeh Mirjafari. Identifying the success factors of communities of practice (cops). *VINE*, 45(2):198–213, 2015.

[AgRa06] Nitin Agarwal and Urvashi Rathod. Defining 'success' for software projects: An exploratory revelation. *INTERNATIONAL JOURNAL OF PROJECT MANAGEMENT*, 24(4):358–370, 2006.

[BFPN17] Tom Bocklisch, Joey Faulkner, Nick Pawlowski, and Alan Nichol. Rasa: Open source language understanding and dialogue management.

[CHW*17] Lei Cui, Shaohan Huang, Furu Wei, Chuanqi Tan, Chaoqun Duan, and Ming Zhou. Superagent: A customer service chatbot for e-commerce websites. In *Proceedings of ACL 2017, System Demonstrations*, Stroudsburg, PA, USA, 2017. Association for Computational Linguistics.

[CuZe05] S. Cummings and A. van Zee. Communities of practice and networks: reviewing two perspectives on social learning. *KM4D Journal*, 1 (1):8–22, 2005.

[CaWh14] Erik Cambria and Bebo White. Jumping nlp curves: A review of natural language processing research [review article]. *IEEE Computational Intelligence Magazine*, 9(2):48–57, 2014.

[CWB*11] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch.

[DeMc92] William Hook DeLone and Ephraim R. McLean. Information systems success: The quest for the dependent variable. *Information Systems Research*, 3(1):60–95, 1992.

[dND*16] Peter de Lange, Petru Nicolaescu, Michael Derntl, Matthias Jarke, and Ralf Klamma. Community application editor: collaborative near real-time modeling and composition of microservice-based web applications. *1617-5468*, 2016.

[FVD*16b] Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. The rise of social bots. *Communications of the ACM*, 59(7):96–104, 2016.

[GKJa08] Anna Glukhova, Ralf Klamma, and Matthias Jarke. Community-aware adaptive systems. In Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, and Christian Scheideler, editors, *INFORMATIK 2008, Beherrschbare Systeme — dank Informatik, Band 2, Beiträge der 38. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 8. – 13. September, in München*, volume 134 of *LNI*, pages 796–801. GI, 2008.

[Hoss19] Phillip Hossner. *Integrating End Users Into Service Success Evaluation Processes.* Master thesis, RWTH Aachen University, Aachen, Germany, 2019.

[JGBM16] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification.

[Kern08] S. J. Kerno. Limitations of communities of practice: A consideration of unresolved issues and difficulties in the approach. In *Journal of Leadership & Organizational Studies*, volume 15, pages 69–78. 2008.

[Kers20] Clemens Kersjes. *Continuous Community Analytics.* Master thesis, RWTH Aachen University, Aachen, Germany, 2020.

[KKK20] Dominik Kus, István Koren, and Ralf Klamma. A link generator for increasing the utility of openapi-to-graphql translations.

[Klam10c] Ralf Klamma. *Social Software and Community Information Systems.* Habilitation, RWTH Aachen University, Aachen, Germany, 2010.

[Klam10e] Ralf Klamma. Werkzeuge und modelle für die übergreifende untersuchung von social software. *i-com*, 9(3):12–20, 2010.

[KlPe08] Ralf Klamma and Zinayida Petrushyna. The troll under the bridge: Data management for huge web science mediabases. In Heinz-Gerd Hegering, A. Lehmann, J. Ohlbach, and C. Scheideler, editors, *Proceedings of the 38. Jahrestagung der Gesellschaft für Informatik e.V. (GI), die INFORMATIK*, pages 923–928. Köllen Druck+Verlag GmbH, Bonn, 2008.

[KRdJ16] Ralf Klamma, Dominik Renzel, Peter de Lange, and Holger Janßen. las2peer – a primer.

[NLKl19] Alexander Tobias Neumann, Peter de Lange, and Ralf Klamma. Collaborative creation and training of social bots in learning communities. In *2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC)*, pages 11–19. IEEE, 12/12/2019 - 12/14/2019.

[Neum18] Alexander Tobias Neumann. *Model-Driven Construction & Utilization of Social Bots for Technology Enhanced Learning.* Masterthesis, RWTH Aachen, Germany, 16.06.2018.

[Renz08] Dominik Renzel. *MobSOS – A Testbed for Mobile Multimedia Community Services.* Diploma thesis, RWTH Aachen University, Aachen and Germany, May 2008.

[Renz16] Dominik Renzel. *Information Systems Success Awareness for Professional Long Tail Communities of Practice.* PhD thesis, RWTH Aachen University, Aachen, Germany, 2016.

[RaKe19] Andrew Rafla and Casey Kennington. Incrementalizing rasa's open-source natural language understanding pipeline.

[RKJa15] Dominik Renzel, Ralf Klamma, and Matthias Jarke. Is success awareness in community-oriented design science research. In Brian Donnellan, Markus Helfert, Jim Kenneally, Debra VanderMeer, Marcus Rothenberger, and Robert Winter, editors, *New Horizons in Design Science: Broadening the Research Agenda*, volume 9073 of *LNCS*, pages 413–420, Switzerland, 2015. Springer International Publishing.

[SaAr05] Sarah Cummings and Arin van Zee. Communities of practice and networks: reviewing two perspectives on social learning. *Knowledge Management for Development Journal*, 1(1):8–22, 2005.

[WCL18] Erik Wittern, Alan Cha, and Jim A. Laredo. Generating graphql-wrappers for rest(-like) apis. In Tommi Mikkonen, Ralf Klamma, and Juan Hernández, editors, *Web Engineering*, volume 10845 of *Lecture Notes in Computer Science*, pages 65–83, Cham, 2018. Springer International Publishing.

[Weng98] Etienne Wenger. *Communities of Practice: Learning, Meaning, and Identity.* Learning in doing. Cambridge University Press, Cambridge, UK, 1998.

[Wies19] Niels Wiessner. *Chat Interfaces for Social Bots in a Peer-to-Peer Environment.* Bachelor, RWTH Aachen, Germany, 2019.

[WWX*16] Yu Wu, Wei Wu, Chen Xing, Ming Zhou, and Zhoujun Li. Sequential matching network: A new architecture for multi-turn response selection in retrieval-based chatbots.

A User Evaluation

A.1 Task Sheet

Task 0 - Joining the Slack workspace

1. Use the following link to join the Slack workspace if you are not a member yet `https://join.slack.com/t/mensacommunity/shared_invite/zt-nf09nfmb-EsChPf76BwKsUUlaR`
2. Use your email address to sign-in or use Google Sign-in
   a) You might need to confirm your email address
   b) Open your email account
   c) You should have received an email from Slack (Also check your spam folder)
   d) Click the link inside the email.
3. You should now be in the Slack workspace
   a) If you have the Slack app, you can click the popup *Open in Slack*
   b) If you don't possess the Slack app, you can also use it in the browser.
4. Locate the bot in the left taskbar near the bottom under Apps
5. You can also search for the bot using the search bar on the top in the middle

Task 1 - Getting to know the bot

Getting the menu

1. Write a greeting message (e.g. Hello)
2. You can type **help** to see a list of the capabilities of the bot
3. Ask the bot to get the menu for your local mensa (canteen)
   - Canteens might be closed
   - You can also ask the bot about canteens in a particular city (e.g what is the menu for mensas in Aachen?)
   - Try using Aachen, Mensa Academica or Aachen, Mensa Vita
4. The bot will look up the menu
5. The bot might ask you if you want to set a default city. Answer with an appropriate message

Make a review

1. Ask the bot to write a review (e.g. I want to add a review)
2. The review process should now be starting
3. The bot will ask you a series of questions
4. First specify which mensa you went to and which meal you had (e.g. I went to Aachen, Mensa Academica and had the Klassiker)
    - The spelling of the category of the dish must be in the same manner as displayed on the menu (**Klassiker, Vegetarisch** not classics or vegetarian)
5. Specify how many stars out of 5 you would give your meal
6. The bot will ask you to leave a comment
    - Leave an appropriate comment
    - You can also type no if you don't want to leave a comment

Task 2 - Make a visualization

This task involves getting insights into the success of the community. Certain success measures are predefined, which can be visualized by using the bot. Visualizations can be either a value, a chart, or a ratio.

1. Ask the bot to make a visualization (e.g Make a visualization)
2. The bot will ask for a measure. Ask the bot to list all measures
3. Choose one of the measures
    - Copy the measure and paste it into the typing field. Then hit ENTER
4. The bot will respond with the appropriate visualization

Task 3 - Update the success model

This task involves updating the success model of the community. The success model is structured into six success **dimensions**. Each dimension contains a list of **factors**. Each factor contains a list of **measures**, which define the visualizations. The success model is based on the requirements of the community. Those requirements were collected during the first evaluation.

1. Use the bot to get the success model (e.g. Get the success model)
2. Aks the bot to update the success model
3. The bot will ask you to choose which success dimension you want to edit

4. Choose a dimension by providing a `number`
5. The bot will now ask you which success factor you want to edit
6. Choose one by providing a `number` or add one by choosing a name for the factor (e.g. Customer Satisfaction)
7. Select one of the measures.
8. The bot will now add the factor to the model.
9. You can also visualize your new measure in the same way as described in Task 2

Feedback

1. Please fill out the following survey: `https://limesurvey.tech4comp.dbis.rwth-aachen.de/index.php/164344?lang=en`
2. If the link does not work, copy-paste the one in the footnotes [1]

A.2 Free text Feedback

”Buttons would be really nice when choosing which visualization to make. When presenting the success model, further text formating would have been nice (e.g. using bold to write the category names). Maybe more nudging in regards to asking users to use the proposed review, visualize and especially success model extension features. Similarly, maybe add some explanations on the benefits and goals of the success model to the bot. Some users might not immediately get the point of the success models. The possibility to change a review or a comment would also be neat. Some visualizations which contained the users' email address in the y-axis resulted in a diagram which was not really lined up with the bars. Overall still a nice and interactive way of asking for the menu and creating communities. Especially when looking at the given use case, one can imagine taking their phone to create such a review or quickly search for the menu before entering the canteen. Especially with the bot being available on one's mobile device and that on a trusted platform. ”

”I like that I can just enter natural language queries to get answers from the bot. It is very good that it always answers quickly. Even if an operation takes longer, it still gives you an initial message to let you know that the bot is still working on providing the answer. I like that the bot has a profile picture. It is very useful to quickly query the menu instead of manually searching it online. Sometimes, I was not 100% sure which kind of answer the bot would understand, so whether I could just talk in free text, enter a number of a list or copy a specific list entry. This aspect could be unified a bit more. To visualize a measure, you have to copy its exact name but to update the

[1]https://limesurvey.tech4comp.dbis.rwth-aachen.de/index.php/164344?lang=en

measure, you can navigate using the numbers. The bot also requires the exact name of the dish category (Klassiker). Here, different alternatives like writing ""classics"" should probably be allowed to increase the user's confidence that the operation is going to succeed. Adding a review is a bit cumbersome as it requires multiple messages with intermediate waiting periods for the bot's answers. Users would probably prefer if they could enter one message like ""I had the Klassiker and would rate it with 5 stars"" to go through multiple steps at once. After that, the bot could just ask for confirmation of all information at once."

"Once, I started with one request, the bot did not find another request, when I wanted to quit the first request and to start a new one. I would have been nice, if the bot immediately change to the new request instead of giving out a error message."

"Sometimes the bot gave an answer right away, even if it just was telling you to wait a second. I liked this very much, as it gave me a very responsive experience. I would wish for the bot to always give such an answer, when having to compute visualizations or the success model, just so I know it heard me and will eventually response."

"Bots are hard to use, in comparison to normal interface, and not fun at all"

"The presentation was well done and was understandable."

"When the bot gave a numbered list of mensas, I entered a number (5), but the bot gave information for another number (2). A lot of listed mensas didn't work, or were not recognised. I had to pay attention how to write commands, e.g. city before mensa, not the other way around. The commands could be more universal. On the plus side, the visualization was good."

"the functionalities seem to be all around well integrated with the chatbot and the intent recognition seems to be very good"

"-clickable list for different tasks.
-favorise ones Mensa, not only cities."

"It's pretty self explainatory, except the success model part."

"There's a general lack of feedback after commands."

A.3 System Usability Scale

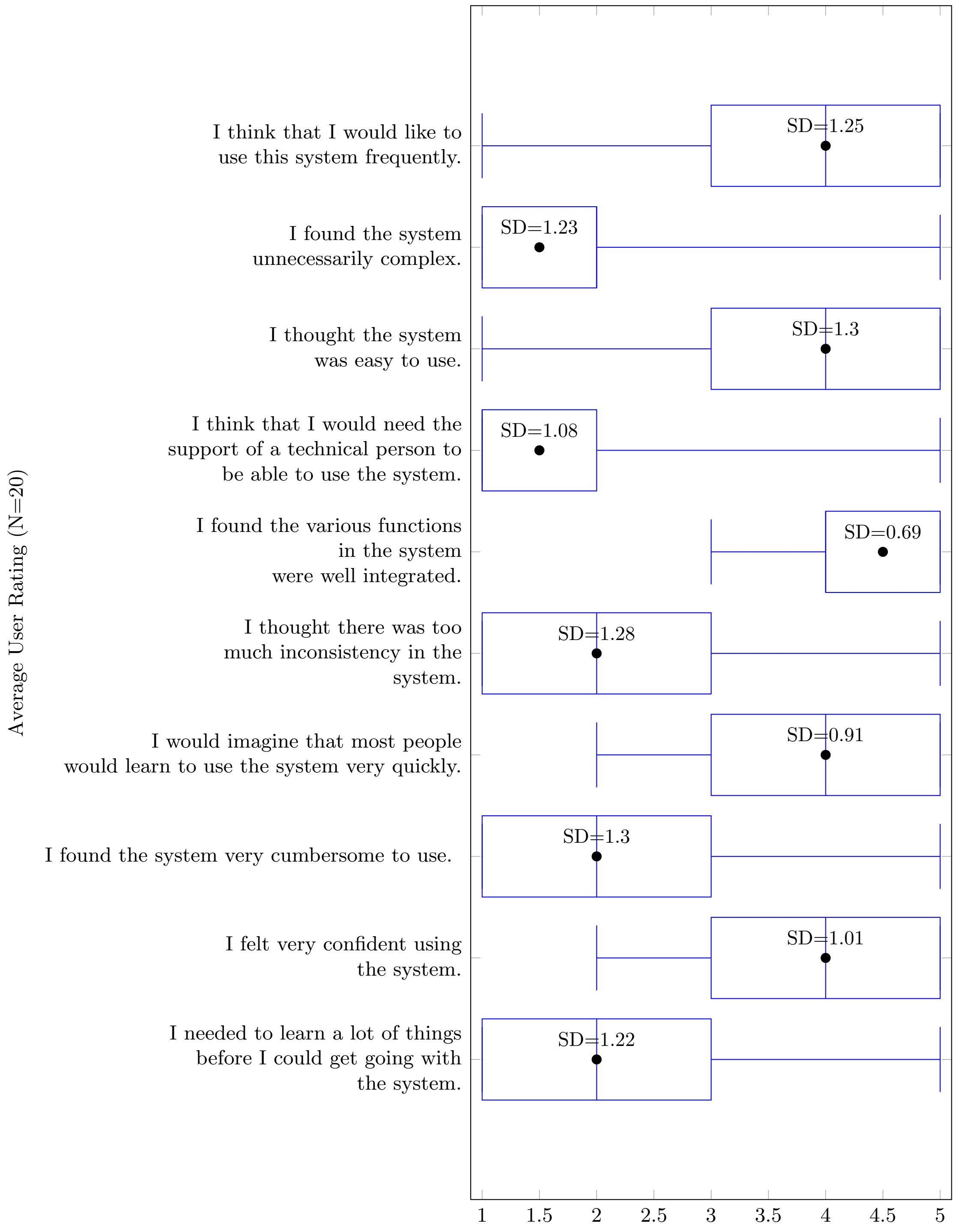


Figure A.1: Results SUS Statements