Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Entwicklung eines Assistenten zur Verbesserung der textbasierten Kommunikation in Entwicklungsteams

Development of an assistant for the improvement of text-based communication in development teams

Masterarbeit

im Studiengang Informatik

von

Niklas Josef Herkenhoff

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüferin: Dr. Jil Ann-Christin Klünder
Betreuer: Alexander Specht, M. Sc.

Hannover, 24. Juli 2023

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 24. Juli 2023

Niklas Josef Herkenhoff

Zusammenfassung

Die vorliegende Arbeit befasst sich mit der Bedeutung der Stimmung für den Erfolg von Softwareprojekten und untersucht, wie ein Kommunikationswerkzeug die positive Stimmung der Nutzer direkt fördern kann. Ein Workshop mit erfahrenen Softwareentwicklern identifizierte Faktoren für negative Stimmung und entwickelte Lösungsansätze auf Seiten der Kommunikationsplattform. Darauf aufbauend wurde ein Konzept für einen Software-Assistenten entwickelt, der die Stimmung live analysiert und bei Bedarf Verbesserungsvorschläge unterbreitet. Der Assistent bietet zudem eine Rechtschreibüberprüfung, berücksichtigt Großschreibung und Interpunktion bei der Stimmungsanalyse und generiert automatische Antwortvorschläge. Verschiedene Ansätze und Technologien zur Implementierung wurden präsentiert, wobei ein Assistenzsystem auf Basis einer Open Source Messenger Software mit KI-Sprachmodellen realisiert wurde.

Um die praktische Anwendbarkeit zu überprüfen, wurde eine Laborstudie durchgeführt, in der Probanden in einem Softwareprojekt-Szenario mithilfe des Prototyps kommunizieren und Programmieraufgaben lösen mussten. Die Ergebnisse zeigen, dass der gewählte Ansatz geeignet ist und der Prototyp als Assistent in einem Entwicklungsteam eingesetzt werden kann. Die Probanden bewerteten die Stimmungsanalysen des Systems als angemessen und die Software wurde aufgrund ihrer guten Usability positiv beurteilt. Die Studie zeigt das Potenzial, die Teamstimmung nachhaltig zu verbessern und unterstützt die praktische Anwendbarkeit des entwickelten Prototyps.

Abstract

Development of an assistant for the improvement of text-based communication in development teams

This paper deals with the importance of sentiment for the success of software projects and examines how a communication tool can directly promote positive user sentiment. At a workshop with experienced software developers, factors leading to poor sentiment were identified and developed approaches to solutions on the part of the communication platform. Based on this, a concept for a software assistant that analyses the sentiment live and makes suggestions for improvement is developed. The assistant also offers a spelling checker, considers capitalisation and punctuation in the sentiment analysis and generates automatic reply suggestions. Various approaches and technologies for implementation are presented, with an assistant system based on open source messenger software using Artificial Intelligence language models.

To test the practical applicability, a laboratory study was conducted in which test persons had to communicate and solve programming tasks in a software project scenario using the prototype. The results show that the chosen approach is suitable and that the prototype can be used as an assistant in a development team. The test persons rated the system's sentiment analyses as appropriate and the software was rated positively due to its good usability. The study shows the potential to sustainably improve team sentiment and supports the practical applicability of the developed prototype.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Komplexe Softwareprojekte können heutzutage nicht mehr nur von einer Person allein bewältigt werden. So gibt es in Softwareunternehmen eine Vielzahl von verschiedenen Mitarbeitenden und Rollen [2]. Diese sind für den Erfolg des Projektes auf einen ständigen Austausch angewiesen [38]. So werden etwa laufend Anforderungen diskutiert, Probleme gemeinsam gelöst oder Wissen ausgetauscht. Die Kommunikation zwischen den Stakeholdern ist daher von entscheidender Bedeutung. Diese findet in Entwicklungsteams zunehmend nicht mehr im direkten Kontakt, sondern online statt [12]. Gerade mit den vielerorts den Mitarbeitenden entgegenkommenden Home-Office-Regelungen und der immer stärker werdenden Verbreitung global verteilter Entwicklungsteams ist die Kommunikation über Plattformen wie Microsoft Teams heutzutage die Regel. [17]

Wie die Kommunikation über diese Plattform gestaltet wird und welches Klima dort herrscht, wird dabei maßgeblich von den Menschen bestimmt. hierbei wird die Bedeutung der sozialen Interaktion innerhalb eines Softwareentwicklungsteams häufig unterschätzt [43]. Denn neben den individuellen fachlichen und sozialen Kompetenzen der Entwicklenden ist auch die allgemeine Stimmung im gesamten Team ein wichtiger Faktor für den Erfolg oder Misserfolg eines Softwareprojekts [44]. So kann eine schlechte Stimmung im Team die Motivation und Produktivität der Mitarbeitenden senken [28].

Gerade in der textbasierten Kommunikation kann die schlechte Stimmung eines einzelnen Mitarbeitenden auf die Stimmung im gesamten Team übertragen. Steht ein/e Mitarbeitende unter Anspannung und schreibt eine negative Nachricht im Affekt, ist diese für Mitarbeitende schwierig einzuordnen. Sie geben möglicherweise eine ähnlich emotionale Reaktion und die Gesamtstimmung verschlechtert sich. Das Fehlen von Mimik und Gestik, die einen wichtigen Teil der Kommunikation ausmachen, erschwert die Einordnung von Nachrichten zusätzlich. Die Ausbreitung negativer

Emotionen von Personen wird auch als emotionale Ansteckung bezeichnet [26].

1.2 Lösungsansatz

Aus diesem Grund wurde ein Lösungsansatz entwickelt, bei dem die Stimmung softwaregestützt über textbasierte Kommunikationskanäle gelenkt werden kann. Hier bietet es sich an, das Potenzial von Instant Messaging Systemen zu untersuchen. Statistiken legen nahe, dass in Zukunft ein immer größerer Anteil der internen Kommunikation in Unternehmen über Plattformen wie Microsoft Teams oder Slack stattfinden wird. E-Mails sollen aufgrund ihres ohnehin eher formellen Charakters nicht berücksichtigt werden. Es zeigt sich auch, dass E-Mails vor allem für die externe Kommunikation genutzt werden. Da der Fokus dieser Arbeit jedoch auf der Stimmung innerhalb von Entwicklungsteams und nicht auf der Stimmung zwischen Kunden und Anbietenden liegt, soll der Fokus auch auf internen Kommunikationsplattformen wie Microsoft Teams liegen. Nicht jedes Unternehmen verwendet jedoch die gleiche Kommunikationsplattform. So werden neben Microsoft Teams auch Programme wie Slack, Skype for Business, Gitter oder Elements [9] eingesetzt. Daher ist es nicht sinnvoll, eine plattformspezifische Lösung zu entwickeln, sondern ein Konzept zu erarbeiten, das auf alle Programme anwendbar ist. Ein solches Konzept für einen Software-Assistenten, der negative Stimmungsübertragung verhindert und positive Stimmung auf Kommunikationsplattformen fördert, soll in dieser Arbeit entwickelt und implementiert werden.

Das grundlegende Konzept für den Assistenten besteht darin, die Stimmung einer gerade verfassten Nachricht mittels Sentimentanalyse zu erkennen und den Nutzenden KI-generierte Handlungsempfehlungen in Abhängigkeit von der erkannten Stimmung anzuzeigen. Neben dem reinen Textinhalt werden in dieser Arbeit weitere Faktoren für negative Stimmung und sinnvolle Handlungsempfehlungen mittels eines Workshops erarbeitet und in der Implementierung der Software berücksichtigt.

Die Sentimentanalyse klassifiziert je nach Ansatz die Stimmung eines beliebig langen Textes in unterschiedlichen Skalen. Meist werden sie in die Klassen *Negativ*, *Neutral* und *Positiv* eingeteilt, aber auch andere Skalen wie eine Sterneanzahl von einem bis fünf Sternen sind üblich. Positive Stimmung kann beispielsweise dadurch gefördert werden, dass der Benutzende auf die Negativität der gerade verfassten Nachricht hingewiesen wird und der Assistent automatisch alternative Formulierungen vorschlägt.

1.3 Ziel der Arbeit

Diese Arbeit setzt sich zum Ziel, einen solchen Prototyp für den Einsatz in Entwicklungsteams zu entwickeln. Dieser Prototyp soll zeigen, welche Möglichkeiten zur Umsetzung es gibt und wie das Konzept gewinnbringend eingesetzt werden kann. Dazu werden folgende Forschungsfragen aufgestellt:

FF1

Welche Einflussfaktoren führen zu negativer Stimmung bei der textbasierten Kommunikation in Entwicklungsteams?

Im ersten Schritt ist es wichtig, die Probleme zu identifizieren, die bei der textbasierten Kommunikation in Entwicklungsteams für negative Stimmung sorgen. Diese sollen als Grundlage für den weiteren Verlauf der Arbeit dienen und Ansatzpunkte schaffen, an denen die Software die Kommunikation unterstützen kann.

FF2

Wie lässt sich ein Assistenzsystem für die Stimmung in einem Entwicklungsteam umsetzen und welche aktuellen Technologien sind dafür geeignet?

Weiter soll geklärt werden, welche technischen Ansätze geeignet sind, um das Ziel des Assistenzsystems zu erfüllen. Dabei soll deutlich gemacht werden, welche Technologien zur Verfügung stehen, zum Beispiel Methoden des maschinellen Lernens aber auch bereits umgesetzte Softwaretools, und wie diese eingesetzt werden können, damit die Software nutzenbringend arbeitet. Dies soll sowohl im Hinblick auf die Umsetzung als auch auf die zu erfüllenden Voraussetzungen geschehen.

FF3

Wie lässt sich das Konzept in der Praxis in Entwicklungsteams einsetzen?

Schließlich soll die Anwendbarkeit des Konzepts in der Praxis untersucht werden. Dabei wird überprüft, ob eine Umsetzung des Systems Einfluss auf die Faktoren zu der Stimmung in Entwicklungsteams nehmen kann und welche Vorteile die Software bringt. Analog dazu werden auch Probleme und Herausforderungen, die der Einsatz mit sich bringt, herausgearbeitet.

1.4 Vorgehensweise

Abbildung 1.1 zeigt die Vorgehensweise der vorliegenden Arbeit. Zur Beantwortung der aufgestellten Forschungsfragen werden im Kapitel Grundlagen

zunächst die für diese Arbeit wichtigen Konzepte wie die Sentimentanalyse und die Funktionsweise von Sprachmodellen erläutert.

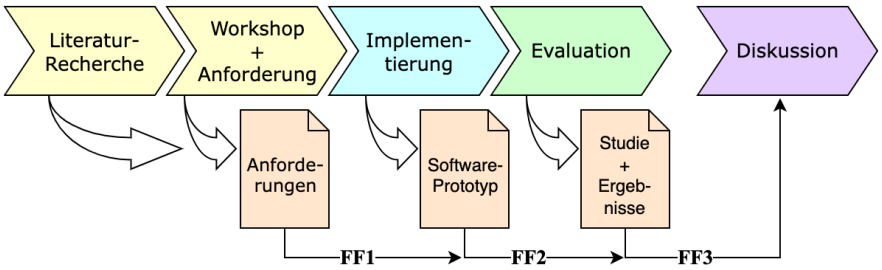


Abbildung 1.1: Übersicht der Vorgehensweise dieser Arbeit

Anschließend werden verwandte Arbeiten vorgestellt und eine Abgrenzung zu diesen vorgenommen, wobei sowohl Unterschiede als auch Gemeinsamkeiten herausgearbeitet werden. Nachdem die Grundlagen dieser Arbeit erläutert wurden, werden im nächsten Kapitel die Anforderungen und der Entwurf des Prototyps behandelt. Dazu wird auch auf die Planung und Durchführung eines Workshops eingegangen. Dieser Teil der Arbeit dient der Beantwortung von **FF1** und bildet somit die Grundlage für die weitere Arbeit. Nach dem Kapitel Implementierung, das sich mit den technischen Aspekten der Umsetzung beschäftigt und damit auf **FF2** eingeht, wird die Durchführung einer Studie zur Evaluation des Prototyps und die daraus resultierenden Ergebnisse zur Beantwortung von **FF3** beschrieben. Abschließend werden die Ergebnisse im Rahmen einer Zusammenfassung und eines Ausblicks in einen größeren Kontext und im Hinblick auf weitere Forschungsmöglichkeiten eingeordnet. Dabei werden zudem die Grenzen dieser Arbeit aufgezeigt und Empfehlungen für zukünftige Forschungsarbeiten formuliert.

Kapitel 2

Grundlagen

Im Folgenden werden die für das Verständnis dieser Arbeit relevanten Grundlagen erläutert. Zunächst soll das Konzept und die verschiedenen Ansätze und Metriken der Sentimentanalyse erläutert werden. Anschließend werden sogenannte *Large Language Models* auf Basis der *Transformer*-Architektur beschrieben. Diese werden in dieser Arbeit sowohl für die Sentimentanalyse als auch für die Formulierung von neuem Text verwendet.

2.1 Sentimentanalyse

Zunächst soll der Begriff der Sentimentanalyse geklärt werden. Dazu wird zunächst erläutert, was unter Sentiment beziehungsweise Stimmung (im Folgenden wird Sentiment und Stimmung analog verwendet) in diesem Kontext zu verstehen ist und welche computergestützten Ansätze es gibt, diese Stimmung zu klassifizieren.

2.1.1 Begriffsklärung: Sentiment

Grundsätzlich gibt es keine allgemeingültige Definition für den Begriff des Sentiments und damit auch nicht für den Forschungsgegenstand der Sentimentanalyse. Ursprünglich stammt der Begriff aus der Linguistik. Quirk grenzt 1985 den sogenannten privaten Zustand (engl. *private state*) als etwas, das einer objektiven Beobachtung oder Überprüfung nicht zugänglich ist, vom sachlichen Inhalt einer Aussage ab [31]. Zum privaten Zustand gehören daher unter anderem Emotionen, Meinungen und Spekulationen. Es muss also unterschieden werden, ob der Forschungsgegenstand eine Meinung, eine Emotion oder das allgemeine Befinden einer Person ist. Im folgenden Abschnitt werden Beispiele für die Anwendung der Sentimentanalyse auf verschiedene Aspekte vorgestellt. Obwohl alle diese Anwendungen in den Bereich der Sentimentanalyse fallen, unterscheiden sie sich durch den betrachteten Aspekt des Sentiments. In dieser Arbeit wird Sentiment explizit

als die Polarität der transportierten Stimmung oder Laune in sprachlicher Interaktion verstanden. Es wird also nur zwischen positivem und negativem Sentiment unterschieden, nicht aber zwischen verschiedenen Emotionen wie *Wut* oder *Trauer*.

2.1.2 Analyse des Sentiments

Die Sentimentanalyse beschreibt die Klassifizierung von Aussagen nach den oben genannten Aspekten des Sentiments. Dies geschieht meist in den Kategorien *Positiv* und *Negativ* oder, wenn die Aussage nicht zugeordnet werden kann, in *Neutral*. Es gibt aber auch Ansätze, die Emotionen feiner zu unterteilen. So kann es nicht nur die Klasse Positiv geben, sondern auch positiv assoziierte Emotionen wie *Freude* oder *Anerkennung* oder auf der anderen Seite des Spektrums *Trauer* oder *Wut*.

Das klassische Anwendungsgebiet ist die Bewertung der öffentlichen Meinung zu einem Thema oder Produkt. So haben Panichella et al. [30] einen Ansatz aufgezeigt, wie die Analyse von App Reviews im Playstore das Entwicklungsteam bei der Pflege und Weiterentwicklung ihrer App unterstützen kann. Eine weitere Möglichkeit ist die Analyse der politischen Stimmung zu bestimmten Themen [32]. Hier wird jeweils die Sentimentanalyse genutzt, um automatisiert Informationen aus großen Datenmengen zu extrahieren und ein umfassendes Stimmungsbild zu erzeugen. Datengrundlage sind hier meist öffentliche Produktrezensionen und Social Media Posts auf Twitter, Facebook und anderen Plattformen.

Im Folgenden sind verschiedene Beispiele zu sehen. Es wird deutlich, dass Sentimentanalysen in verschiedenen Kontexten eingesetzt werden könne. Zudem kann die Analyse Sätze neben *Negativ* und *Positiv* auch in verschiedenste andere Klassen kategorisieren.

Beispiel für Sentimentanalyse

„Der Service war super.“ → Positiv

„Die App läuft ganz schön langsam.“ → Negativ

„Das ist doch echt nicht zu fassen!“ → Wut

„Das Gerät ist für den Preis ganz in Ordnung“ → 3/5 Sternen

Ein weiterer Anwendungspunkt ist die Analyse der Kommunikation in Open Source Projekten wie beispielsweise auf GitHub[1]. Ziel war es herauszufinden, ob sich Methoden der Sentimentanalyse auch auf Software Engineering Prozesse anwenden lassen [18].

[1] https://github.com/explore | zuletzt aufgerufen am: 09.06.2023

2.1.3 Methoden der Sentimentanalyse

Die Stimmungsanalysetools lassen sich grundsätzlich in zwei verschiedene Ansätze unterteilen. Zum einen gibt es die lexikonbasierten Tools. Bei diesen werden die Aussagen anhand von positiven und negativen Wörtern kategorisiert. Das Lexikon besteht aus einer großen Liste von Grundformen der Wörter, denen jeweils ein positiver oder negativer Score zugeordnet ist. Die Summe der Scores aller Wörter in einem Textausschnitt bestimmt dann die Gesamtstimmung. Je nach Implementierung werden dabei auch negierende Wörter wie „nicht“ berücksichtigt, so dass das nachfolgende Wort in der Bewertung umgekehrt wird. Analog dazu können auch verstärkende Wörter wie „sehr“ die Bewertung ins Positive oder Negative verstärken. Im folgenden Unterkapitel wird eine ausgewählte Implementierung dieses Verfahrens genauer vorgestellt.

Der zweite Ansatz ist die Verwendung von Methoden des maschinellen Lernens. Im Gegensatz zu lexikonbasierten Werkzeugen werden dem Klassifikationssystem keine Regeln vorgegeben, sondern es erlernt diese eigenständig. Dazu wird in den meisten Fällen eine Datengrundlage benötigt, mit der das System trainiert wird. Diese besteht aus Eingabedaten, die mit einem Label versehen sind. Eine solche Eingabe kann je nach Art der Klassifikation aus einem Wort, einem Satz oder auch einem ganzen Textdokument bestehen. Diese Eingabesätze werden jeweils einer Klasse zugeordnet (im Beispiel der Stimmungsanalyse sind dies *negativ*, *neutral* oder *positiv*). Die Eingabetexte werden dann in einem für den Computer zu verarbeitenden Vektor überführt. Dieser Vektor kann insofern abstrahiert werden, als er nicht nur aus den rohen Zeichenketten besteht, sondern jedem Wort z.B. die Auftrittshäufigkeit oder semantische Informationen (wie die Positionierung in einem Wortnetz, in dem semantisch ähnliche Wörter, z.B. Synonyme, nahe beieinander liegen) zugeordnet werden. Somit wird bei der Klassifikation die Bedeutung von Wörtern über die reine Buchstabenfolge (Syntax) hinaus berücksichtigt. Da es sich bei der Sentimentanalyse um ein klassisches Klassifikationsproblem handelt, können dementsprechend auch klassische Methoden wie *Support Vector Machines* (SVM) oder *Naive Bayes* angewendet werden.

Darüber hinaus werden auch neuronale Netze und Deep Learning eingesetzt. Dabei befinden sich zwischen Eingabe und Ausgabe sogenannte *Hidden Layer*, die wiederum aus Neuronen bestehen. Jedes *Neuron* nimmt eine Eingabe und verarbeitet sie zu einer Ausgabe weiter. Durch die Anzahl der Schichten und die hohe Vernetzung der Neuronen können auch komplexere Zusammenhänge herausgearbeitet werden.

2.2 Vorstellung SentiStrength

Zunächst wird eine Implementierung mit einem lexikonbasierten Ansatz vorgestellt. Dabei werden insbesondere die Nachteile herausgestellt, die

in den nächsten Abschnitten den Methoden des maschinellen Lernens gegenübergestellt werden.

SentiStrength ist ein Stimmungsanalyse-Tool, das von Thelwall an der University of Wolverhampton [36] entwickelt wurde. Es folgt dem oben beschriebenen lexikonbasierten Ansatz und ist für kurze Texte aus dem sozialen Web gedacht. Dabei wird für jedes Wort, aus dem der Satz besteht, in einer Tabelle abgefragt, ob es eine positive oder negative Polarität ausdrückt. Negative Sentiments werden mit Werten von -1 (nicht negativ) bis -5 (extrem negativ) erfasst, positive Sentiments entsprechend mit Werten von 1 (nicht positiv) bis 5 (extrem positiv). Dabei können Begriffe durch bestimmte Schlüsselwörter wie „nicht“ oder „sehr“ in ihrer Bewertung negiert oder verstärkt werden. Grundsätzlich behandelt *SentiStrength* die Parameter für positive und negative Wörter getrennt. Dadurch kann das Ergebnis je nach gewähltem Ausgabeformat unterschiedlich dargestellt werden. Beispielsweise kann das Ergebnis das Tupel aus dem Wert für das negativste Wort und dem Wert für das positivste Wort sein. Andererseits kann das Ergebnis auch die Summe aller positiven und negativen Polaritäten sein.

Während dieser Ansatz für einfache Beispiele mit eindeutig einer Polarität zuordenbarem Vokabular angemessen und zuverlässig funktioniert, schneidet die Methode bei komplexeren Satzstrukturen schlechter ab.

Beispiel 1

„Deine Arbeit war super[+3] schnell fertig und echt gut[+2][+1 booster word] umgesetzt.“
(Max Positiv: 3 und Max Negativ: -1): Positiv

Beispiel 1 zeigt die Funktionsweise von *SentiStrength* anhand eines einfachen Beispiels. Hier wurden die beiden positiven Wörter „super“ und „gut“ (zusammen mit dem Steigerungswort „echt“) im zugrundeliegenden Lexikon gefunden. Der maximale Wert gibt dabei die Punktzahl für Positiv an. Die Punktzahl für Negativ ist bei -1, da kein negatives Wort gefunden wurde und daher der Standardwert übernommen wurde.

Beispiel 2

„Deine Umsetzung ist der Hammer, und das obwohl es bestimmt echt beschwerlich[-2][-1 booster word] war.“
(Max Positiv: 1 und Max Negativ: -3): Negativ

Im zweiten Beispiel werden die Probleme dieser Methode deutlich. Die Aussage, welche die Anerkennung ausdrückt, dass jemand eine Aufgabe gut erledigt hat, auch wenn der Weg dorthin schwierig war, wird von *SentiStrength* negativ eingeordnet. Obwohl die meisten Menschen den Satz positiv bewerten würden, wird dies nicht erkannt, da etwa „etwas ist der

Hammer“ im Lexikon nicht als positiv hinterlegt ist. Außerdem wird nicht berücksichtigt, dass der vermeintlich negative Teil des Satzes dazu dient, die ausgedrückte Anerkennung zu verstärken, anstatt Kritik oder Ähnliches zu äußern. Dieses Beispiel zeigt, dass ein solcher Lösungsansatz stark von der verwendeten Datenbank und den manuell eingegebenen Regeln abhängt, um die Tonalität eines Satzes richtig zu interpretieren.

2.3 Transformer

Zur Lösung von Problemen im Bereich des *Natural Language Processing* (NLP), zu dem auch die Sentimentanalyse gehört, werden zunehmend KI-Modelle eingesetzt. Unter *Natural Language Processing* wird die Verarbeitung menschlicher Sprache mit Hilfe von Regeln aus dem traditionellen Computing, dem maschinellen Lernen und *Deep-Learning*-Modellen verstanden. Das Ziel besteht darin, dass Computer lernen, den semantischen Inhalt von Text oder gesprochenen Wörtern zu verstehen und zu verarbeiten. NLP-Probleme umfassen die Übersetzung von Texten aus einer Sprache in eine andere, die Generierung von Antworten in Chatbots, das Erkennen von Eigennamen in Texten und vieles weitere. Aber auch das Klassifizieren von Textelementen wie in der Sentimentanalyse und das Umformulieren von Sätzen fallen in diesen Bereich. Aus diesem Grund befasst sich das folgende Kapitel mit dem aktuellen Stand der Technik im *Natural Language Processing*. Besonderes Augenmerk wird dabei auf Sprachmodelle aus dem Fachgebiet des *Deep Learnings* gelegt, da Sprachmodelle dieser Art in der Implementierung der Software dieser Arbeit verwendet werden.

2.3.1 Entwicklung der Transformer-Architektur

Der älteste und einfachste Ansatz für ein solches Sprachmodell ist ein *Probalistic Model*. Ein solches Modell kann genutzt werden, um Texte zu klassifizieren. Beispielsweise können so E-Mails als Spam oder auch die Stimmung in *Positiv* oder *Negativ* eingeteilt werden.

Dabei wird anhand der Häufigkeit bestimmter Wörter eine Voraussage getroffen. Dabei werden ausschließlich Wörter berücksichtigt, die in einem vordefinierten Vokabular enthalten sind. Kommt ein Wort in diesem so genannten *Bag of Words* vor, bestimmt das trainierte Gewicht, wie stark sich das Auftreten auf die Wahrscheinlichkeit der Voraussage auswirkt. Das Training wird auf Basis eines gelabelten Trainingsdatensatzes durchgeführt [45].

Dieser Ansatz nutzt bereits wichtige Konzepte wie die Kodierung von natürlichsprachlichem Text, die Vorhersage anhand von Wahrscheinlichkeitswerten bei diskreten Entscheidungen und den Einsatz von gelabelten Datensätzen. Allerdings zeigt dieser Ansatz ein Problem auf. Die Methode kann für einfache Aufgaben wie die Erkennung von Spam-E-Mails

zuverlässig eingesetzt werden, zeigt aber bei komplexeren Problemen wie der Stimmungsanalyse eine geringere Performanz. Entscheidend dafür ist, dass die Reihenfolge, in der die Wörter stehen, vernachlässigt wird. Auch Negationen wie „nicht“ würden bei dieser Methode ignoriert und ihre Bedeutung vernachlässigt werden.

Eine Weiterentwicklung der einfachen *Probalistic Models* sind die sogenannten *N-Gram-Modelle*. Hierbei wird für jedes Wort auch die Einbettung in die Wörter vor dem Wort mit berücksichtigt. Diese Wortblöcke werden zusammengefasst und als einzelne Entitäten betrachtet. Diese sogenannten *N-Gramme* (N steht für die Größe des betrachteten Bereichs) werden anhand ihrer Auftrittshäufigkeit in den Trainingsdaten bewertet. Durch die Berücksichtigung der Umgebung eines Wortes können *N-Gramm-Modelle* für weiterführende NLP-Probleme eingesetzt werden. Zum Beispiel kann Text automatisch generiert werden, indem die Vorhersagewahrscheinlichkeit jedes generierten Elements durch die vorhergehenden Elemente bestimmt wird. Auch die grammatikalische Plausibilität kann berechnet werden, da das sehr seltene Auftreten bestimmter n-Gramme auf grammatikalische Fehler hindeuten kann [45].

Allerdings geht auch dieser Ansatz von naiven Annahmen aus. Es lassen sich nämlich leicht Beispiele finden, bei denen Wörter in wechselseitiger Abhängigkeit zueinander stehen, aber nicht unmittelbar benachbart sind.

Dieses Problem nimmt sich die Benutzung von *Recurrent Neural Networks* (RNNs) an. Bei der Verarbeitung von Texten werden die Wörter sequentiell eingelesen, die jeweils den internen Zustand, der rekursiv auch die Information über die bisherigen Wörter enthält. Anhand des Zustands kann am Ende der Sequenz zum Beispiel das nächste Wort vorhergesagt oder eine Klassifikation durchgeführt werden. Dieses Vorgehen hat neben der Annahme, dass die semantischen Abhängigkeiten der Wörter zum aktuellen Wort linear mit der Nähe zum aktuellen Wort zunehmen, auch das Problem, dass das Sprachmodell konzeptbedingt keinen Vorteil aus der Parallelisierung ziehen kann [45].

Nachdem die grundlegenden Konzepte von Sprachmodellen erläutert wurden, wird im Folgenden auf das *Transformer*-Modell eingegangen. Dieses ist für diese Arbeit insofern von Bedeutung, als dass die implementierte Software an mehreren Stellen auf Sprachmodelle dieses Typs zurückgreift.

Die grundlegende Idee von *Transformer*-Modellen besteht darin, die semantische Repräsentation eines Wortes in Abhängigkeit von allen anderen Wörtern des Satzes zu modellieren. Dabei sind diese Abhängigkeiten nicht wie bei *N-Gram-Modellen* konstant, sondern je nach Stärke der Abhängigkeit gewichtet.

Die variablen Abhängigkeiten werden mit der sogenannten *self-attention*-Technik [39] erreicht. *Self-attention* beschreibt das Erlernen der individuellen Gewichtung der Beziehung zwischen jeder Entität einer Eingabe mit jeder

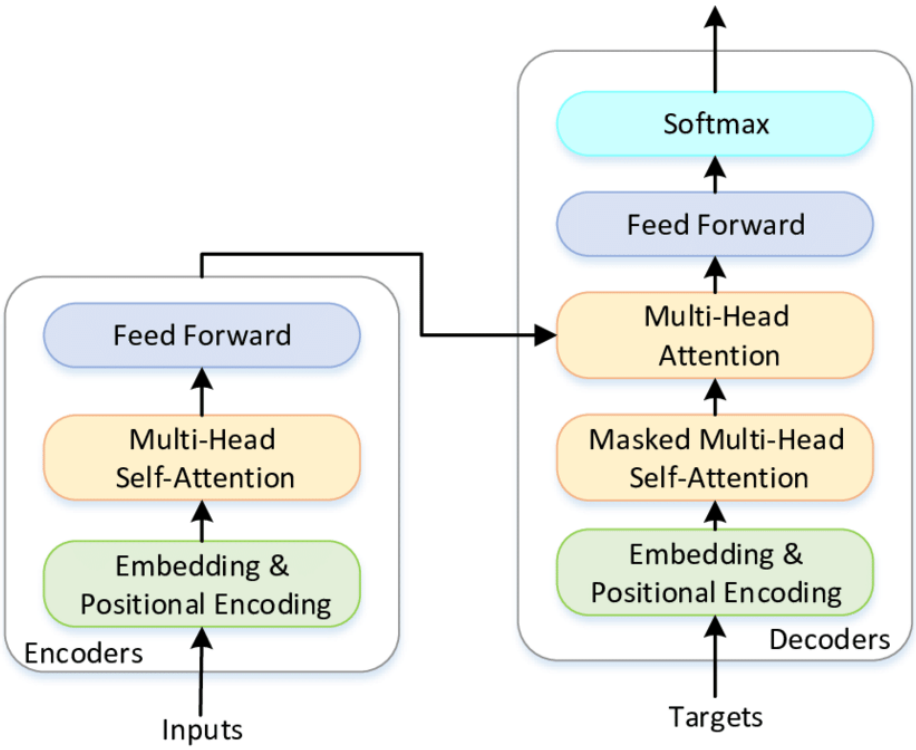


Abbildung 2.1: Eine vereinfachte Darstellung der Architektur von Transformer Modellen (aus Anwar et al. [4])

anderen Entität derselben Eingabe. Dadurch, dass Eingaben stets als Ganzes betrachtet werden, können Kontexte und semantische Abhängigkeiten besser verstanden werden.

Die gesamte Architektur besteht aus einer Enkoder-Dekoder-Struktur (siehe Abbildung 2.1). Der Enkoder überträgt die Informationen der Eingabe in einen abstrakten Vektorraum. Diese Ergebnisse werden in den Dekoder übertragen. Hier im Dekoder werden nacheinander einzelne Token generiert, die selber wiederum analog zu dem Enkoder im Dekoder weiter verarbeitet werden [4][45]. Neu generierte Ausgabe wird demnach bei der weiteren Generierung direkt mit berücksichtigt.

Abbildung 2.2 zeigt die unterschiedlichen Abhängigkeiten und Beziehungen für die Interpretation der einzelnen Wörter in den verschiedenen Modellen. Hier ist zu erkennen, dass die Beziehung des Wortes „es“ im *N-Gramm-Modell* (4-Gramm) nur zu den vier Wörtern vor dem Auftreten betrachtet wird. Allerdings wird das Wort „Umsetzung“ nicht in die Betrachtung einbezogen, obwohl es sich in der Bedeutung des Satzes auf dieses bezieht. Ebenso wird beispielsweise das Adjektiv, welches das „es“ beziehungsweise die „Umsetzung“ beschreibt („beschwerlich“), nicht mit einbezogen.

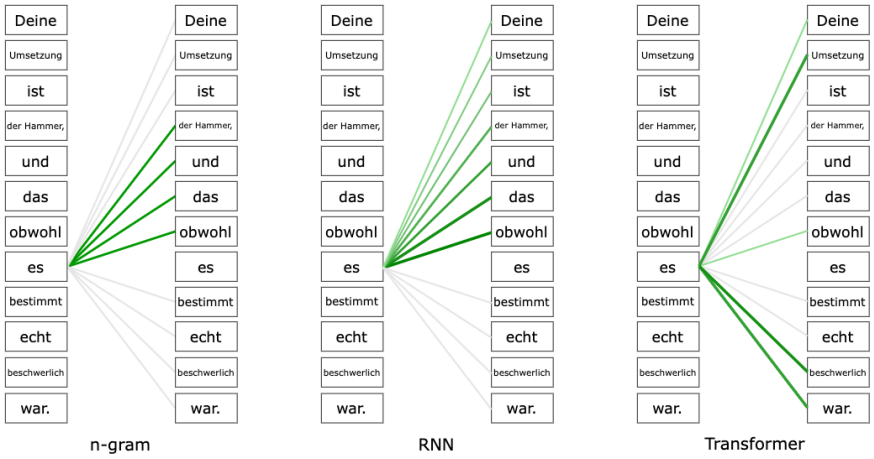


Abbildung 2.2: Ein Vergleich der Abhängigkeiten in der Repräsentation von Wörtern in den verschiedenen Architekturen

Während das RNN alle Wörter vor dem „es“ mit linear aufsteigendem Gewicht in die Interpretation einbezieht, zeigt sich, dass die *self-attention* des *Transformer*-Modells die Bedeutung am realistischsten abbilden kann. Es werden sowohl die Beziehungen vor als auch nach dem Auftreten berücksichtigt. Dieses kurze Beispiel dient dazu, die Entwicklung in der computergestützten Verarbeitung natürlicher Sprache zu verdeutlichen. Es soll auch zeigen, warum *Transformer*-basierte Verfahren im Vergleich zu ihren Vorgängern sehr erfolgreich sind, die semantische Bedeutung und Aussage von Sprache zu „verstehen“.

Neben der Betrachtung von sogenannten Fernabhängigkeiten (engl. *Long-range Dependencies*), die durch *self-attention* ermöglicht werden, hat das Design von *Transformer*-Modellen weitere Vorteile. So lassen sich Modelle, die auf der *Transformer*-Architektur basieren, besser parallelisieren als RNNs. Dies wird, wie oben beschrieben, durch die auf Rekursion basierende Architektur der *Recurrent Neural Networks* verhindert. Somit können Modelle nicht nur mit einem besseren Sprachverständnis, sondern durch die Parallelisierung auch mit mehr Datensätzen und damit mehr Wissen trainiert und betrieben werden.

2.3.2 BERT

Im Jahr 2018 wurde das Transfomer-basierte Machine-Learning-Modell BERT entwickelt. BERT steht für „Bidirectional Encoder Representations from Transformers“. „Bidirectional“ steht für die Verwendung des *Self-Attention*-Mechanismus, der, wie oben erläutert, die Modellierung von Ab-

hängigkeiten vor und nach dem Wort ermöglicht. Es hat sich in verschiedenen Benchmarks als leistungsfähiger für NLP-Probleme erwiesen als bisherige Modelle. Im Folgenden soll geklärt werden, wie die Klassifizierung, zum Beispiel der Stimmung, mit dem Sprachmodell BERT funktioniert.

Die BERT-Architektur besteht aus einem Stapel mehrerer Encoder. Jeder Encoder besteht aus einer *self-attention* und einer *feed forward* Schicht. Die *feed forward* Schicht ist ein simples künstliches neuronales Netzwerk ohne Rekursion, wie etwa bei einem RNN. Das in dieser Arbeit genutzte Modell basiert auf BERT base. Es besteht aus 12 Schichten von Enkodern und hat somit 110 Millionen trainierbare Parameter.

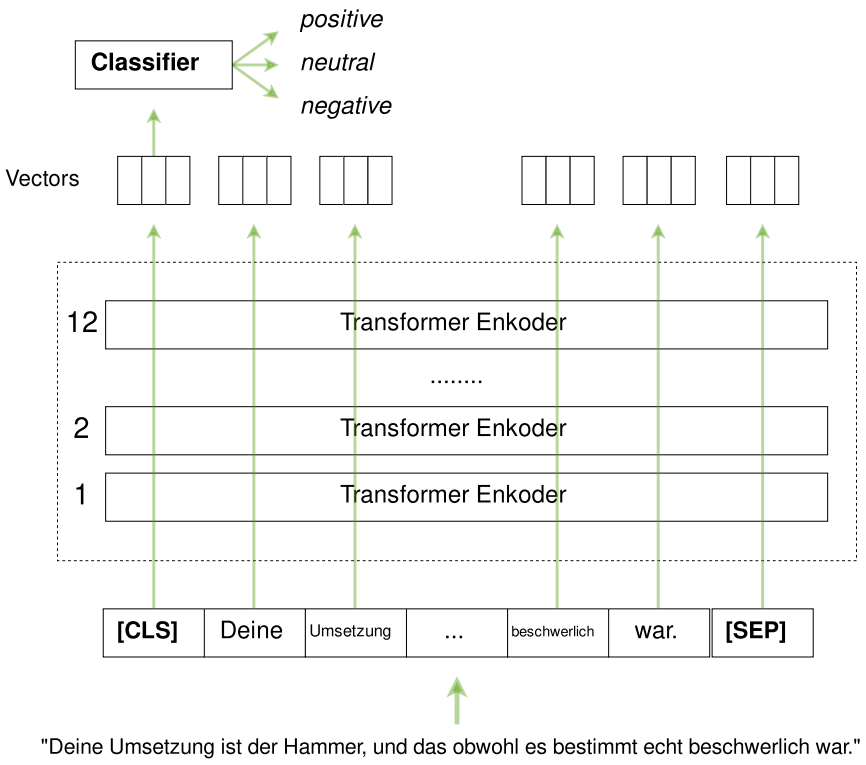


Abbildung 2.3: Vereinfachte Funktionsweise eines BERT Klassifizierungs Modells

Abbildung 2.3 skizziert die Funktionsweise eines BERT-Klassifikationsmodells. Ein BERT-Modell erwartet eine Folge von Wörtern, die als *Token* bezeichnet werden. Für ein Klassifizierungsproblem benötigt die Eingabe zusätzlich zwei spezielle *Token*. Zum einen ist dies das [CLS], das Klassifizierungstoken, welches am Anfang der Sequenz stehen muss und

zum anderen das [SEP]. Dieses *Token* grenzt Eingaben voneinander ab. Wird nur ein Satz als Eingabe genutzt, steht dieses *Token* am Ende der Eingabe. Diese Sequenz an *Token* wird nun von den Enkoderschichten in Vektoren übertragen. Für die Klassifizierung wird lediglich die Vektorrepräsentation des Klassifizierungstokens [CLS] benötigt. Diese verfügt aufgrund des *self-attention*-Mechanismus über genügend Informationen über den Kontext des Satzes, um mittels eines auf den Vektor angewendeten Klassifikators eine Vorhersage zu treffen. Dieser Klassifikator, der das [CLS]-Token klassifiziert, besteht aus einer trainierten Gewichtsmatrix und einer Aktivierungsfunktion [13].

2.3.3 German Sentiment Classification Model

Das in dieser Arbeit genutzte Modell zur Klassifizierung von Stimmungen wurde von Guhr et al. [13] im Jahr 2020 entwickelt. Ziel war es, ein allgemeingültiges Modell für die deutsche Sprache zu entwickeln, das in den unterschiedlichsten Kontexten und Domänen eingesetzt werden kann. Dabei konzentriert sich der Zweck dieses BERT-Modells insbesondere auf die Klassifikation in einer Dialogumgebung. Das Modell wurde mit verschiedenen Datensätzen aus unterschiedlichen Domänen trainiert. Dazu gehörten beispielsweise von Experten publizierte, manuell annotierte Datensätze wie PotTS [34] oder GermEval-2017 [42]. Diese Datensätze wurden von jeweils zwei Experten manuell annotiert und umfassen unter anderem politische Diskussionen, Social Media, Blogs, alltägliche Konversationen. Ergänzt wurden diese Datensätze mit Bewertungen und Rezensionen aus verschiedenen Bereichen sowie deutschen Auszügen aus der Wikipedia. Insgesamt wurden 1,834 Millionen Datenpunkte für das Training genutzt, nachdem Datenpunkte aussortiert wurden, um die Häufigkeit der drei Klassen *Negativ*, *Neutral* und *Positiv* anzugleichen. In der Arbeit wurde gezeigt, dass ein BERT-Modell sowohl in einer bekannten Domäne als auch insbesondere in einer unbekannten Domäne bessere Ergebnisse liefert als ein Vergleich mit einem *N-Gramm*-basierten Ansatz. Wegen der Fokussierung auf Dialogsysteme und der hohen Flexibilität in den Domänen wurde dieses Modell für die Stimmungsklassifikation in dieser Arbeit verwendet.

Um einen Einblick in die Funktionsweise der Klassifizierung des obigen Beispielsatzes zu erhalten, wurden mit Hilfe des *model explainability* Tools *Transformers-Interpret*[2] die Gewichte für das Ergebnis der Vorhersage visualisiert.

Abbildung 2.4 zeigt diese Gewichte. Die grüne Farbe bedeutet, dass ein Token stark zu der gewählten Klassifizierung (*positiv*) beiträgt, die rote Farbe, dass es die Klassifizierung abschwächt. Zunächst fällt auf, dass einzelne Wörter auch in mehrere Token zerlegt werden können. Bei der

[2] https://github.com/cdpierse/transformers-interpret | zuletzt aufgerufen am: 09.06.2023

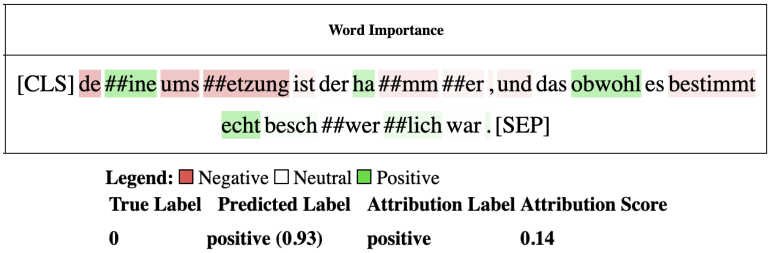


Abbildung 2.4: Gewichte für das Positiv Label generiert von *Transformers-Interpret*

Bewertung ist zu erkennen, dass der Hauptsatz eher gemischt bewertet wird. Allerdings trägt der Nebensatz, der durch *SentiStrength* noch negativ bewertet wurde, nun positiv zur Bewertung bei. Insbesondere das Wort „obwohl“ gefolgt von der Beschreibung der Herausforderung hat einen großen Anteil an der positiven Gesamtbewertung. Dieses Beispiel zeigt, dass das BERT-Modell die Sätze auf einer tieferen Ebene betrachtet als zum Beispiel *SentiStrength*.

2.3.4 LLaMA

Das LLaMA Modell von Meta AI zählt zu den *Large Language* Modellen (LLM). Diese künstlichen neuronalen Netzwerkmodelle zeichnen sich dadurch aus, dass sie für die Verarbeitung natürlicher Sprache und insbesondere für die Generierung von Texten entwickelt wurden und dass eine enorm große Anzahl an trainierbaren Parametern im Netz vorhanden ist. Je größer die Anzahl an Parametern ist, desto komplexere Muster kann das Sprachmodell erkennen und generieren. Mit der Entwicklung der *Transformer*-Architektur hat sich die Leistungsfähigkeit solcher Modelle stark verbessert. Der Vorteil von großen vortrainierten Sprachmodellen besteht darin, dass sie ohne weiteres Training für eine Vielzahl von NLP-Aufgaben eingesetzt werden können. Durch sogenanntes *Transfer Learning* können erlernte allgemeine Sprachmuster genutzt und auf eine Vielzahl von Anwendungen übertragen werden. Durch die große Menge an Trainingsdaten und das damit erworbene Sprachverständnis kann ein *general purpose Large Language* Modell selbst NLP-Aufgaben zum Teil besser lösen als hoch spezialisierte aber kleinere Sprachmodelle [37].

LLaMA bietet verschiedene Modellgrößen von 7 Milliarden bis zu 65 Milliarden trainierbaren Parametern. Im Vergleich zu dem sehr weit verbreiteten Modell GPT3 von OpenAI [7] wurde bei der Entwicklung mehr Wert auf die Größe des Trainingsdatensatzes gelegt, anstatt eine bessere Performance durch eine Erhöhung der trainierbaren Parameter zu erreichen.

Dies führt dazu, dass das Training eines Modells zunächst kostenintensiver ist, die Nutzungskosten hinterher jedoch sinken. Die Trainingsdaten umfassen unter anderem Webseiten, Open-Source-Projekte aus Github, Wikipedia in zwanzig verschiedenen Sprachen, öffentlich zugängliche Literatur, LaTeX-Quellcode von wissenschaftlichen Arbeiten und Fragen und Antworten aus Foren wie etwa StackExchange. Durch die Verfügbarkeit der trainierten Gewichte gibt es viele Open-Source-Projekte, die das vortrainierte Modell weiter trainieren, um beispielsweise die Interaktion mit dem Sprachmodell zu vereinfachen [46].

Kapitel 3

Verwandte Arbeiten

In Kapitel 2 wurden bereits die grundlegenden Konzepte und Anwendungsbereiche der Sentimentanalyse vorgestellt. Dieses Kapitel beschäftigt sich mit Forschungsarbeiten, in denen die Sentimentanalyse speziell im Bereich Software Engineering eingesetzt wird. Anschließend werden Forschungen und Projekte aus dem Bereich der Schreibassistenten vorgestellt. Zuletzt wird eine Abschlussarbeit von Schroth beleuchtet und die vorliegende Arbeit von dieser und anderen Forschungsarbeiten abgegrenzt.

3.1 Sentimentanalyse im Software Engineering

Neben den im Grundlagenkapitel vorgestellten Anwendungsbereichen der Sentimentanalyse gibt es auch im Bereich Software Engineering eine Vielzahl von Forschungsarbeiten, die sich mit dem Einsatz der Sentimentanalyse beschäftigen. So haben Obaidi und Klünder [29] in ihrer systematischen Literaturrecherche aus dem Jahr 2021 insgesamt 6763 Arbeiten zu diesem Thema identifiziert und ausgewertet. Nach dem Aussortieren von Duplikaten und der inhaltlichen Filterung wurden schließlich 80 Arbeiten näher betrachtet. Hierbei sind 59 Publikationen im Bereich der Open Source Software Projekte angesiedelt. Dahinter beschäftigen sich 17 Publikationen mit dem Einsatz in der Industrie und 6 mit dem Bereich Bildung.

Es hat sich gezeigt, dass die meisten Probleme, die in diesen Arbeiten genannt wurden, die Anpassung der Werkzeuge an die Domäne des Software-Engineerings betreffen. Insgesamt wurde dies in 26 der 80 Arbeiten als Problem erwähnt. Um dieses Problem zu lösen, wurde mit *SentiStrength-SE*, eine Weiterentwicklung von *SentiStrength*, 2017 das ursprüngliche Tool um ein domänenspezifisches Vokabular und angepasste Heuristiken erweitert [16]. Islam et al. [16] haben in einer Studie gezeigt, dass domänenspezifische Systeme bessere Ergebnisse liefern als Allgemeine.

Auch präsentieren Colefato et al. [8] mit Senti4SD ein Sentimentanalysetool für die Domäne Software Engineering. Hierbei werden lexikonbasierte,

schlüsselwortbasierte und semantische Merkmale genutzt um Stimmung in einer Polarität darzustellen. Diese semantischen Merkmale werden mittels eines neuronalen Netzwerkes extrahiert. Verfahren des Maschinellen Lernens können demnach auch mit den traditionellen Methoden kombiniert werden. Ihr Ansatz mit einem kleinen Trainingsdatensatz bereits eine bessere Performance als *SentiStrength*. *Senti4SD* ist nur für die englische Sprache verfügbar.

Mit dem Einsatz der Sentimentanalyse zur Verbesserung des sozialen Miteinanders in Entwicklungsteams beschäftigen sich unter anderem Guzman und Bruegge [14]. Sie stellen einen Ansatz vor, wie ein emotionales Bewusstsein in Entwicklungsteams geschaffen werden kann. In ihrem Ansatz werden die ausgedrückten Emotionen automatisch aus allen kollaborativen Artefakten extrahiert und zusammengefasst. Die Extraktion erfolgt über das oben vorgestellte *SentiStrength*. Zusätzlich werden die Ergebnisse, unabhängig von der Verwendung als Monitoring, wie in der vorliegenden Arbeit, gewinnbringend weiterverarbeitet. Dabei werden die Werte von *SentiStrength* mittels Themenmodellierung verschiedenen inhaltlichen Themen zugeordnet und anschließend zusammengeführt. Dieser Ansatz wurde in drei verschiedenen Entwicklungsteams über einen Zeitraum von drei Monaten angewendet. Anschließende Interviews mit den Teamleitungen zeigten, dass die Zusammenfassungen der Emotionen gut mit dem tatsächlichen emotionalen Status der Mitarbeitenden des Projekts korrelierten.

3.2 Schreibassistenten

Seit dem Aufkommen des Forschungsbereichs Natural Language Processing gibt es eine Vielzahl von Ansätzen, um das Schreiben von Texten digital zu unterstützen. Diese decken dabei sowohl wissenschaftliche als auch kommerzielle Kontexte ab. Dale und Viethen [10] haben 2021 die aktuellen Forschungen und Tools zusammengetragen. Dabei beleuchten sie auch die Historie und gegenwärtige Trends bei computergestützten Schreibassistenzsystemen. Traditionell adressieren Schreibassistenten drei Probleme bei schriftlichen Texten: Rechtschreib-, Grammatik- und Stilüberprüfung. Während Rechtschreib- und Grammatiküberprüfung relativ einfach zu definieren sind, nämlich als eine Überprüfung im Hinblick auf die allgemein als korrekt akzeptierte Abfolge von Buchstaben und Wörtern, geht die Stilüberprüfung tiefer als die generelle Syntax. Die Stilüberprüfung beschäftigt sich neben den Konventionen der Rechtschreibung und Grammatik mit der Art und Weise, wie ein bestimmter Inhalt ausgedrückt werden kann. Hierbei fließen neben dem zur Verfügung stehenden Wortschatz auch der beabsichtigte kommunikative Zweck, die Zielgruppe und die persönliche Beziehung zwischen den Beteiligten mit ein. Hier gestaltet sich die Aufstellung von Regeln schwieriger als beispielsweise bei der Rechtschreibüberprüfung. So gab es in

dem Feld bereits 1961 eine Anwendung der Rechtschreibüberprüfung, die von Earnest [23] entwickelt wurde und auf einer einfachen Liste von korrekten Wörtern beruhte. Jedes Wort, welches nicht in der Liste auftaucht, wird als Fehler deklariert. Später wurden auch der Kontext des Wortes und das aus dem falsch geschriebenen Wort hervorgehende wahrscheinlichste Wort mit einbezogen. Dadurch konnten schon früh gute Ergebnisse erzielt werden. Ähnlich verhält es sich mit Grammatiküberprüfern.

Stilüberprüfungssysteme, zu denen auch der in dieser Arbeit vorgestellte Prototyp zählt, existieren seit den 1980er Jahren. So gibt es beispielsweise den RightWriter[1] und den Stylewriter[2]. Diese Werkzeuge zielen darauf ab, das Schreiben professioneller zu gestalten und werden hauptsächlich von Schriftstellenden und redaktionell Tätigen genutzt. Technisch basieren diese Assistenten auf manuell erstellten Regeln, die spezifische Muster in Texten erkennen und ebenfalls manuell erstellte Erklärungen und Verbesserungsvorschläge bereitstellen. Neben dem Einsatz für Schriftstellende haben sich im Laufe der Zeit weitere Anwendungsgebiete und Regelsätze entwickelt. Beispielsweise konzentriert sich PerfectIt[3] auf technische Formulierungen und WorkRake[4] auf die Formulierung juristischer Texte.

Technisch ausgefeilter sind dagegen die Marktführer im Bereich der Schreibassistenten Grammarly[5] und LanguageTool[6], welches für die deutsche Sprache geeignet ist. Beide bieten eine umfassende Rechtschreib- und Grammatiküberprüfung und unterbreiten direkt eine verbesserte Version der Eingabe mit Erklärungen. Zusätzlich bieten sie KI-gestützte Umformulierungen an. Dabei zielen sie darauf ab, die Formalität der geschriebenen Sätze zu erhöhen. Bei beiden Tools ist jedoch erkennbar, dass sie nicht für die Forschung, sondern für die Industrie entwickelt wurden. So sind die Algorithmen und Sprachmodelle, mit denen die Verbesserungen generiert werden, nicht öffentlich zugänglich. Eine Ausnahme bildet hier LanguageTool, bei dem die Grundfunktionalitäten der Rechtschreib- und Grammatiküberprüfung Open Source sind, nicht jedoch die Stilüberprüfung.

3.3 Abschlussarbeit von Schroth

In seiner Abschlussarbeit entwickelt Schroth ein Konzept, um die Stimmung in einem Entwicklungsteam in Echtzeit zu erfassen [33], mit dem Ziel negativer Stimmung vorzubeugen. In einer prototypischen Implementierung

[1] https://www.right-writer.com | zuletzt aufgerufen am: 09.06.2023
[2] https://www.editorsoftware.com | zuletzt aufgerufen am: 09.06.2023
[3] https://intelligentediting.com | zuletzt aufgerufen am: 09.06.2023
[4] https://wordrake.com | zuletzt aufgerufen am: 09.06.2023
[5] https://grammarly.com | zuletzt aufgerufen am: 09.06.2023
[6] https://languagetool.org | zuletzt aufgerufen am: 09.06.2023

setzt er dazu das oben vorgestellte *SentiStrength* in Kombination mit einem sogenannten *Keylogger* ein, um die Stimmung in der schriftlichen Kommunikation zu analysieren. *SentiStrength* wird dabei in zwei Ausführungen verwendet. Zum einen die deutsche und aufgrund der häufigen Verwendung von Anglizismen in Entwicklungsteams die englische Version. Außerdem wird die Nutzung von Emoticons und Soziolekten berücksichtigt. Soziolekte sind dabei wiederkehrende Ausdrücke aus dem Sprachgebrauch der Nutzenden, die von der Analyse durch *SentiStrength* nicht erfasst werden. Beispielsweise würde der Ausdruck „etw. ist der Hammer“ aus dem obigen Beispiel darunter fallen. Durch den *Keylogger* werden alle Tastatureingaben des Nutzenden erfasst und die Daten können bei jedem Teammitglied dezentral gesammelt werden. Wenn beispielsweise ein Chatprogramm eine Nachricht geschrieben wird, wird das standardisierte Ergebnis der Sentimentanalyse der Nachricht auf einer Skala von -1 bis 1 (Kommabereich mit eingeschlossen) angezeigt. Dies sorgt bei den Nutzenden für ein Bewusstsein über die transportierte Stimmung der eigenen Nachricht. Zusätzlich wird eine Begründung beziehungsweise die Berechnung des Ergebnisses angezeigt. Durch eine anonymisierte Weitergabe der Ergebnisse kann ein Gesamtdurchschnitt berechnet werden, der die allgemeine Stimmung im gesamten Team abbildet. In einer Studie mit zwölf Teilnehmenden konnte gezeigt werden, dass die Stimmung sinnvoll in Echtzeit zu erfassen ist und auf dieser Grundlage abschätzbar wird, wann präventive Maßnahmen zur Verbesserung der Stimmung notwendig sind.

3.4 Abgrenzung der Arbeit

Die vorgestellten Arbeiten zur Sentimentanalyse verfolgen alle den Ansatz, die Ergebnisse der Sentimentanalyse außenstehend zu analysieren oder den Anwendenden beziehungsweise der Teamleitung zur Verfügung zu stellen. Dabei wird die Interpretation der Ergebnisse und die Schlussfolgerungen aus den Ergebnissen nicht automatisch gezogen. Den Nutzenden bleibt es in diesen Beispielen überlassen, ob und vor allem wie sie auf die negative Stimmung reagieren und Maßnahmen ergreifen. Der praktische Nutzen ist also stets nur indirekt gegeben.

Diese Arbeit hat zum Ziel, die Ergebnisse der Sentimentanalyse um einen unmittelbaren praktischen Nutzen zu erweitern. Dazu wird der Ansatz der Live-Sentimentanalyse dahingehend ergänzt, dass negative Stimmungen nicht nur aufgezeigt und visualisiert werden. Sie werden darüber hinaus zum Anlass genommen, negative Nachrichten automatisch umzuformulieren. Dieser Ansatz lässt sich dem Bereich der Schreibassistenten genauer der Stilüberprüfung zuordnen. Hierbei unterscheidet sich das Vorgehen in dieser Arbeit insofern von den genannten Tools, dass nicht nur die Formulierung,

sondern auch der Inhalt beeinflusst wird. Während LanguageTool und andere vor allem einen formellen Stil fördern, indem Wörter und Kollokationen durch formellere ersetzt werden, interpretiert der Prototyp dieser Arbeit den Inhalt des Satzes und formuliert ihn in einer freundlicheren Weise um. Diese Neuformulierung verschiebt den semantischen Inhalt bewusst in die positive Richtung. Dies steht im Gegensatz zu den oben beschriebenen Werkzeugen, die den Text in seiner Bedeutung möglichst unangetastet lassen.

Trotzdem lässt sich der Prototyp dieser Arbeit von der Benutzeroberfläche und der Funktionsweise von Tools wie LanguageTool und Grammarly inspirieren. Der Prototyp und die Ähnlichkeiten werden in den folgenden Kapiteln genauer beschrieben. Im Anschluss daran werden die technischen Aspekte der Implementierung beleuchtet.

Kapitel 4

Konzeptentwicklung

Nachdem in Kapitel 2 die technischen Grundlagen erläutert und in Kapitel 3 die Abgrenzung zu bestehenden Arbeiten vorgenommen wurde, soll nun die Entwicklung des Konzepts erläutert werden. Dazu wird zunächst dargestellt, welche Vorstellungen und Erwartungen an die Software gestellt werden. So werden neben einer Vision, die das Ziel der Software definiert, auch Szenarien entwickelt, bei denen die entwickelte Software unterstützen soll. Danach wird ein Workshop beschrieben, in dem Probleme bei der schriftlichen Kommunikation in Entwicklungsteams identifiziert und Lösungsansätze diskutiert wurden. Zuletzt werden weitere Anforderungen an die Software definiert und ein Entwurf vorgestellt.

4.1 Vision

Kollmann et al. [21] haben untersucht, welchen Einfluss es auf das Design von Softwaresystemen hat, die Bedürfnisse der Endnutzenden zu erfassen. Dabei wurde festgestellt, dass eine konkrete Vision des Nutzens einer Software im Designprozess von entscheidender Bedeutung ist. Eine Vision sollte die Ziele aus technischer, geschäftlicher Perspektive und vor allem die gewünschte Benutzererfahrung definieren. Das Definieren einer Vision kann auch die Kommunikation mit Stakeholdern erleichtern. Dies wurde auch in dieser Arbeit durch den durchgeführten Workshop zur Identifizierung von Problemen gezeigt, die sich auf die Stimmung in Softwareentwicklungsteams auswirken.

Die Formulierung einer Vision, das sogenannte *Vision Statement*, soll bewusst eine langfristige und auch idealisierte Zukunftsvorstellung beschreiben. Sie soll aufzeigen, in welcher Weise das Produkt den Menschen helfen kann und wie sich dies auf das Leben der Menschen auswirkt. Dabei muss nicht berücksichtigt werden, inwieweit die Vision in der Praxis erreicht werden kann. Die Schritte und Maßnahmen zur Erreichung der Vision sind nicht Teil des *Vision Statements*. Ziel ist es lediglich, ein gemeinsames

Verständnis über die Intention eines (Software-)Produktes zu schaffen [20].

Vision Statement

›In Zukunft werden Assistenzsysteme für Kommunikationsplattformen im geschäftlichen Bereich nicht mehr wegzudenken sein. Durch die fortschreitende Globalisierung und die Entwicklung hin zu neuen, flexibleren Arbeitsformen werden Teams noch häufiger als heute verteilt arbeiten. Das bedeutet, dass das Bild von Teams, die gemeinsam in einem Büro sitzen, zunehmend der Vergangenheit angehören wird. Dies gilt insbesondere für Entwicklungsteams in der IT-Branche, aber auch für viele andere Branchen, in denen nicht direkt handwerklich gearbeitet wird. Die in die Kommunikationsplattformen eingebetteten Assistenten werden neben der Rechtschreibprüfung noch viele weitere Quality-of-Life-Verbesserungen beinhalten. So wird es neben der Zusammenfassung verpasster Nachrichten und der Motivation des Assistenten, schnell auf Nachrichten zu antworten, auch Stimmungsanalysen geben, die verhindern, dass Verfassende missverständliche Nachrichten an ihre Teammitglieder schreiben. Der Assistent formuliert negative Nachrichten automatisch so um, dass keine Missverständnisse entstehen können und negative Stimmungen nicht im Team verbreitet werden. Außerdem müssen die Mitarbeitenden nicht mehr lange auf Antworten auf ihre Nachrichten warten, da jeder Mitarbeitende von der Software motiviert wird, schnell, freundlich und informativ zu antworten. So wird negative Stimmung vermieden und positive Stimmung gefördert. Langfristig wird sich die Diskussionskultur verbessern, so dass eine rege Kommunikation und eine positive, wertschätzende Ausdrucksweise zum neuen Standard werden. Die Mitarbeitenden können sich besser auf ihre eigentliche Aufgabe konzentrieren, da sie weniger Zeit damit verbringen, über Formulierungen nachzudenken und sich den Kopf über die Bedeutung negativer Nachrichten von Teammitgliedern zu zerbrechen. So kann auch in stressigen Phasen ein angenehmes Arbeitsklima gewährleistet und gleichzeitig der Stressfaktor und die Unzufriedenheit der Mitarbeitenden reduziert werden.‹

4.2 Workshop

Um Anforderungen und Funktionalitäten neben der Sentimentanalyse von Nachrichten zu definieren, wurde im Rahmen dieser Arbeit ein Workshop geplant und durchgeführt. Ziel des Workshops war es, Probleme zu identifizieren, die sich auf die Stimmung bei der Nutzung von Kommunikationsplattformen wie Microsoft Teams auswirken. Dabei wurden auch Lösungsansätze diskutiert, die auf Seiten der Kommunikationssoftware

realisiert werden können. Der Workshop soll zusammengefasst die folgenden Leitfragen beantworten:

1. Welche Probleme treten in der alltäglichen Nutzung von Kommunikationstools auf?
2. Welchen Einfluss haben diese Probleme auf die Stimmung der Mitarbeitenden?
3. Wie kann die Kommunikationssoftware selbst dabei helfen, die Probleme zu lösen oder zu vermeiden?

4.2.1 Planung des Workshops

Zu dem Workshop wurden vier Personen eingeladen. Es wurde dabei darauf geachtet, dass die Teilnehmer des Workshops im Bereich der Softwareentwicklung tätig sind und regelmäßig mit Kommunikationsplattformen wie etwa Microsoft Teams arbeiten. Dies ist für den Workshop insofern wichtig, als dass die Teilnehmer aus ihrer individuellen Erfahrung Probleme herausarbeiten sollen. Ein technisches Verständnis für die Entwicklung von Software ist ebenfalls von Vorteil, um die Umsetzbarkeit von Lösungen einschätzen zu können. Der Workshop soll in Präsenz stattfinden, damit verschiedene Werkzeuge wie Partnerarbeit, Brainstorming und Mindmapping reibungslos umgesetzt werden können. Außerdem soll eine lebhafte Atmosphäre für Diskussionen geschaffen werden, so dass die Ideen aller Teilnehmer erfasst werden können. Zusätzlich kann bei Anwesenheit besser beobachtet werden, inwieweit Probleme nur individuell auftreten oder ob die Zustimmung auch von anderen Teilnehmenden kommt. Dies wäre beispielsweise bei einem Workshop über ein Videokonferenzsystem schwieriger. Zeitlich ist der Workshop auf etwa zwei Stunden ausgelegt.

4.2.2 Durchführung des Workshops

Wie oben beschrieben, nahmen vier Personen neben dem Autor an dem Workshop teil. Davon sind drei Personen in der Softwareentwicklung tätig. Eine Person hat gerade ihren Master in Informatik abgeschlossen. Alle Teilnehmer sind männlich und haben sowohl mehrere Jahre Erfahrung in der Softwareentwicklung als auch im Umgang mit Kommunikationsplattformen im beruflichen oder akademischen Kontext. Im Durchschnitt sind die Teilnehmer 29 Jahre alt. Der Workshop fand in einem Konferenzraum statt, der mit einem großen Tisch in der Mitte, einem Whiteboard mit entsprechenden Stiften und einer großen Leinwand für Präsentationen ausgestattet war. Nach der Begrüßung fand ein kurzes Kennenlernspiel statt, um direkt zum Anfang eine lockere Atmosphäre für die Teilnehmer zu schaffen und den Austausch anzuregen. Anschließend wurde das Thema Sentimentanalyse

und die Ausgangssituation der Arbeit erläutert. Die Ausgangssituation bestand zu diesem Zeitpunkt aus der Idee, die Sentimentanalyse in eine bestehende Kommunikationsplattform zu integrieren. Auch die zugrunde liegende Kommunikationssoftware, in der die Sentimentanalyse eingesetzt werden soll, wurde grob vorgestellt (mehr dazu unter Kapitel 5), um ein gemeinsames Verständnis zu schaffen. Als erste Aufgabe wurde im Plenum ein Brainstorming durchgeführt, bei dem jeder Probleme nennen konnte (siehe Leitfragen 1 und 2). Diese wurden auf einem Whiteboard notiert. Dabei wurde darauf geachtet, dass ähnliche und doppelte Wortmeldungen unter einem Oberbegriff zusammengefasst werden. Dies wurde so lange fortgesetzt, bis keine Neuen mehr gefunden wurden. Im nächsten Schritt wurden die Teilnehmer in Kleingruppen aus zwei Personen eingeteilt. Diese Gruppen arbeiteten dann in Partnerarbeit daran, ausgehend von den im ersten Schritt identifizierten Problemen , Lösungsansätze zu finden und diese schriftlich auf einem Blatt Papier festzuhalten (siehe Leitfrage 3). Dabei war es wichtig, dass die Teilnehmer nur Lösungsansätze entwickelten, die durch die Software bzw. den Assistenten in der Software umgesetzt werden können. Lösungsansätze, die sich auf das soziale Miteinander in den Entwicklungsteams beziehen, sollten nicht ausgearbeitet werden, da hier nur die Möglichkeiten der Software untersucht werden sollten. Die gefundenen Lösungsansätze wurden am Ende wieder im Plenum gesammelt und auf ihre Umsetzbarkeit hin diskutiert. Die Ergebnisse aus diesem Schritt wurden von der Workshopleitung schriftlich gesammelt. Am Ende des Workshops wurden die Stichpunkte (identifizierte Probleme) an der Tafel abfotografiert und die Notizen der Partnerarbeit eingesammelt (siehe Anhang A). Diese dienen als Grundlage für die weitere Auswertung.

4.2.3 Ergebnisse des Workshops

Abbildung 4.1 zeigt die genannten Aspekte, die negative Auswirkungen auf die Stimmung der Teilnehmenden im Kontext der schriftlichen Kommunikation haben. Es wurden ähnliche Nennungen unter neuen Oberbegriffen zusammengefasst. Die neuen Oberbegriffe wurden ebenfalls im Workshop diskutiert. Es wurde stets darauf geachtet, dass alle Teilnehmer ihre Nennungen in den Oberbegriffen wiederfinden beziehungsweise ihnen zuordnen können. Dieses Vorgehen diente dem Ziel, die Anzahl der Aspekte überschaubar zu halten, ohne dass dabei aber Aspekte verloren gehen. Die einzelnen Aspekte werden im Folgenden genauer erklärt.

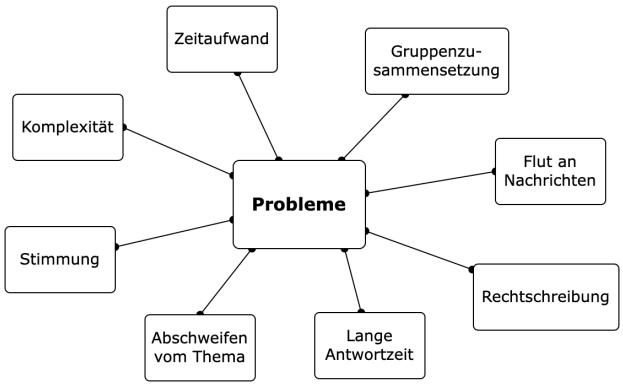


Abbildung 4.1: Erarbeitete Mindmap mit den Ergebnissen der Problemidentifizierung

Ergebnisse Leitfrage 1 und 2

a) Stimmung: Die Teilnehmer berichteten, dass sie sich gestört fühlen, wenn sie mit Personen kommunizieren, bei denen sie an der Art der Nachrichten merken, dass sie schlecht gelaunt sind. Neben der reinen Ausdrucksweise (unfreundliche Sprache) werden auch die Verwendung bzw. das Fehlen von Emoticons und die Länge der Nachrichten genannt. Längere Nachrichten werden von den Teilnehmern bevorzugt, da sie mehr Informationen enthalten. Zu kurze Nachrichten wirken auf die Teilnehmer oft unfreundlich.

b) Abschweifen vom Thema: Dieser Punkt bezieht sich darauf, dass in öffentlichen Chats häufig Dinge diskutiert werden, die nicht für alle Teilnehmer relevant sind. So kommt es häufig vor, dass in einer Gruppe Sachverhalte diskutiert werden, die nicht mehr zum anfänglichen Thema passen und besser in einem anderen Rahmen geklärt werden sollten.

c) Rechtschreibung: Es stört die Teilnehmenden des Workshops, wenn sie Nachrichten mit vielen Rechtschreib- oder Grammatikfehlern empfangen. Dies wirkt sich insofern auf die Stimmung aus, dass das Lesen und Verstehen der Nachrichten somit mehr Aufmerksamkeit und Mühe erfordert. Auch kann es zu einem Mehraufwand für die Beteiligten kommen, wenn aus diesem Grund Rückfragen entstehen und geklärt werden müssen.

d) Lange Antwortzeit: Die Teilnehmenden nannten zu lange Antwortzeiten auf ihre Fragen. So sei es frustrierend, wenn eine Anfrage in einem Gruppenchat oder an bestimmte Personen erst nach langer Zeit oder gar nicht beantwortet wird. Oftmals wird erst nach nochmaligem Nachhaken

geantwortet. Diese Frustration führe im schlimmsten Fall dazu, dass immer weniger Fragen gestellt würden, da man bereits davon ausgehe, keine Antwort zu bekommen. Dies kann je nach Situation schwerwiegende Folgen haben, wenn zum Beispiel Unklarheiten bei Anforderungen nicht geklärt werden, sondern vom Entwickelnden nach eigenem Belieben umgesetzt werden.

e) Komplexität: Nach Angaben der Teilnehmer werden in den Gruppenchats häufig Themen diskutiert, die für eine schriftliche Kommunikation zu komplex sind. Dabei handelt es sich beispielsweise um technische Aspekte und Umsetzungsprobleme, aber auch um Diskussionen zwischen zwei oder mehreren Personen, die sich nicht einig zu sein scheinen. Solche Themen können zum Beispiel besser in einer Telefon-/Videokonferenz besprochen werden, da hier die schriftliche Kommunikation an ihre Grenzen stößt.

f) Flut an Nachrichten: Dieser Aspekt befasst sich mit dem Erhalt von übermäßig vielen Nachrichten innerhalb eines kurzen Zeitraums. Die Teilnehmer beschreiben es als störend, wenn sie von einer Person mehrere Nachrichten hintereinander erhalten, da ihr Arbeitsfluss dadurch immer wieder unterbrochen wird. Auch die schnelle Abfolge der Benachrichtigungstöne wird als lästig empfunden.

g) Gruppenzusammensetzung: Themen werden oft in größeren Gruppen besprochen, obwohl sie nicht für alle relevant sind. Dies führt dazu, dass Personen, für die die Nachrichten uninteressant sind, dem Chat weniger Aufmerksamkeit schenken und so möglicherweise wichtige Nachrichten verpassen.

h) Zeitaufwand: Die Teilnehmer empfinden die Kommunikation über Messengersoftware als hohen Zeitaufwand, der sie von ihrer primären Entwicklungsarbeit ablenkt.

Wie oben beschrieben, wurden zu den einzelnen Problemen auch Lösungsansätze diskutiert. Die Art der Ansätze fällt dabei unterschiedlich aus. So gibt es Aspekte, bei denen sich die Teilnehmer eingestehen mussten, dass ihre Lösungsideen in der Umsetzung eher unrealistisch sind.

Ergebnisse Leitfrage 3

a) Stimmung: Die Teilnehmer schlagen vor, Nachrichten vor dem Versenden farblich zu hinterlegen. Negative Nachrichten könnten rot hinterlegt werden, um bei dem Absendenden ein Bewusstsein für die Stimmung zu schaffen, die durch die Nachricht hervorgerufen werden kann. Analog dazu können positive

Nachrichten grün hinterlegt werden. Dies könnte dazu führen, dass bei jeder Nachricht das Unterbewusstsein angesprochen wird und beim Erscheinen der grünen Farbe ein Gefühl der Zufriedenheit ausgelöst wird. Auf diese Weise könnten negative Stimmungen in den Nachrichten vermieden und positive gefördert werden. Darüber hinaus wurde vorgeschlagen, dass negative Sätze durch künstliche Intelligenz automatisch in einen vergleichsweise positiveren Satz umformuliert werden könnten.

b) Abschweifen vom Thema: Hier gab es den Vorschlag, den Chat so aufzubauen, dass zu einem neuen Thema ein neuer Thread erstellt wird. Antwortet eine Person auf einen solchen Thread, wird die Antwort direkt bei dem Thread angezeigt und nicht am Ende des Chats. Auf diese Weise werden die Nachrichten nach Themen sortiert und nicht nach dem Zeitpunkt, an dem sie gesendet wurden. Um diese Funktion zu automatisieren, wurden Methoden der künstlichen Intelligenz vorgeschlagen. Auch wurde die Idee einer Suchfunktion für Nachrichten geäußert.

c) Rechtschreibung: Hier wurde neben der klassischen Autokorrektur auch das Editieren von Nachrichten nach dem Absenden genannt. Die Anwendung könnte automatisch eine Bearbeitung vorschlagen, wenn sie davon ausgeht, dass die nutzende Person die vorangegangene Nachricht bearbeiten möchte.

d) Lange Antwortzeit: Die Teilnehmer erwähnten die stark verbreitete Funktion, Personen in Nachrichten zu markieren. Häufig wird dies durch ein @-Symbol vor dem Namen realisiert. Die Markierung einer Person führt dazu, dass sich die Person, von der eine Antwort gewünscht wird, direkt angesprochen fühlt und die Hemmschwelle, eine Nachricht zu ignorieren, sinkt. Zusätzlich wurde die Idee entwickelt, Nachrichten je nach Dringlichkeit unterschiedlich zu behandeln. Die Erkennung der Dringlichkeit kann manuell oder automatisch erfolgen. Es wurde auch vorgeschlagen, dem Empfänger automatisch Antwortmöglichkeiten oder Textbausteine für die Antwort anzubieten, um den Aufwand für die Antwort zu minimieren.

e) Komplexität: Ein System, das anhand der Menge der Nachrichten in einem kurzen Zeitraum erkennt, ob die schriftliche Kommunikation angemessen ist. Ist dies nicht der Fall, schlägt es dem Benutzer vor, ein Telefongespräch oder eine Videokonferenz mit den beteiligten Personen zu führen.

f) Flut an Nachrichten: Eine kurze Zeitspanne nach dem Versenden einer Nachricht, in der der Benutzer keine Nachrichten mehr verfassen kann, kann hier Abhilfe schaffen. Analog zum Aspekt der Rechtschreibung wurde hier der Lösungsansatz des Editierens von Nachrichten aufgegriffen. Wird

eine Nachricht editiert und die Korrektur nicht in einer weiteren Nachricht vorgenommen, reduziert sich die Anzahl der Nachrichten insgesamt.

g) Gruppenzusammensetzung: Zu diesem Aspekt konnte kein Lösungsansatz erarbeitet werden. Es wurde lediglich ein Verweis auf die Möglichkeiten von KI gegeben, nicht aber geklärt wie KI konkret helfen kann.

h) Zeitaufwand: Analog zum Aspekt der langen Antwortzeiten wurde auch hier überlegt, inwieweit der Aufwand für das Verfassen von Nachrichten reduziert werden kann. Die Hemmschwelle wird beispielsweise durch das automatische Vorschlagen von Antwortmöglichkeiten gesenkt. So kann der Benutzer bequem mit der Maus eine Nachricht auswählen, ohne zur Tastatur greifen zu müssen.

Diskussion der Ergebnisse

Nachdem nun alle Probleme und Lösungsansätze aus dem Workshop erfasst sind, werden im Folgenden die Lösungsansätze bewertet. Dabei fällt auf, dass einige der Lösungsansätze so oder in ähnlicher Form bereits in verbreiteten Kollaborationswerkzeugen vorhanden sind. So wird beispielsweise die Organisation von Nachrichten in Threads von Microsoft Teams und Slack umgesetzt. Neben einer Suchfunktion für Nachrichten existiert in diesen Anwendungen auch ein System zum Markieren von Personen. Da diese Funktionen bereits branchentypisch sind, werden sie in dieser Arbeit nicht weiter untersucht.

Darüber hinaus werden Lösungsansätze vernachlässigt, die nicht durch ein Assistenzsystem innerhalb eines ansonsten herkömmlichen Kommunikationstools umgesetzt werden können und eine weitergehende Umstrukturierung der gesamten Plattform erfordern. Beispielsweise würde eine Neuorganisation des Nachrichtenverlaufs nach Themen oder die Komprimierung aller Nachrichten in Zusammenfassungen eine tiefgreifende Anpassung des Systems erfordern. So müsste etwa die Datenbank der Nachrichten neu strukturiert werden. Dieses Vorgehen wäre nicht im Rahmen der technischen Möglichkeiten eines Browser-Plugins. Ziel des Prototyps ist es wie oben erläutert, ein Assistenzsystem darzustellen, welches neben dem Beispiel-Messenger auch in verschiedene andere Tools, wie z.B. Microsoft Teams, integriert werden könnte.

Ausgehend von diesen Überlegungen wurden die folgenden Funktionen für die Umsetzung des Prototyps festgelegt.

1. **Live-Sentimentanalyse der Eingabe mitsamt farblicher Hervorhebung des Textes**

2. **Automatische Neuformulierung von Sätzen mit negativen Sentiment**
3. **Positive Antwortvorschläge beim Empfang einer Nachricht**
4. **Hinweis bei einer zu großen Längendifferenz zwischen Frage und Antwort**
5. **Automatisiertes Bearbeiten einer bereits gesendeten Nachricht nach zeitnaher Folgenachricht**
6. **Rechtschreibkorrektur**

Hauptfokus soll in dieser Arbeit auf den ersten beiden Punkten liegen. Die genaue Funktionsweise wird in den Anwendungsszenarien und im User Interface dieses Kapitels und in Kapitel 5 Implementierung dargestellt.

4.3 Weitere Anforderungen

Nachdem die grundlegenden Funktionen des Prototyps herausgearbeitet wurden, werden in diesem Unterkapitel bewährte Verfahren für das Design von Oberflächen von Empfehlungssystemen (engl. Recommendation Systems) behandelt. Murphy und Murphy-Hill [24] haben fünf Faktoren aufgestellt, die bei dem Design von Empfehlungssystemen beziehungsweise Software-Assistenten berücksichtigt werden sollen: Verständlichkeit, Transparenz, Zugänglichkeit, Vertrauen und Ablenkung. Sie gehen davon aus, dass damit Empfehlungen und Hinweise in einer Software den Nutzenden einen Mehrwert bieten, das Interface dementsprechend gestaltet sein muss.

Verständlichkeit

Nutzende müssen in der Lage sein, die Empfehlung des Systems zu erkennen und den Hinweis zu verstehen. Es gibt zwei Dimensionen, in denen die Verständlichkeit ausgedrückt wird: Offensichtlichkeit und kognitive Belastung. Offensichtlichkeit beschreibt, inwieweit die Nutzenden ohne Vorkenntnisse erkennen können, dass ein System eine Empfehlung ausspricht. Hier spielt insbesondere der Wiedererkennungswert der Darstellung eine entscheidende Rolle. Die kognitive Belastung beschreibt, inwieweit es dem Nutzer möglich ist, die Empfehlung beiläufig zu erkennen und zu verstehen. Wenn Nutzende einen Hinweis nicht direkt erkennen und einen Aufwand betreiben müssen, um ihn zu finden, wird die Empfehlung nicht gewinnbringend eingesetzt.

Um Empfehlungen verständlich zu konzipieren, sollten die Prinzipien der Mensch-Computer-Interaktion beachtet werden. Dabei gelten für die Gestaltung von Empfehlungen insbesondere „Übereinstimmung von System und Realität“, „Konsistenz und Standards“ und „Hilfe und Dokumentation“

[27]. Es sollten Metaphern verwendet werden, die die Nutzenden aus der realen Welt kennen. Ähnliche Ereignisse sollten so gestaltet werden, dass Nutzende sie wiedererkennen. Das letzte Prinzip „Hilfe und Dokumentation“ beschreibt, dass das System grundsätzlich ohne Hilfe bedient werden kann, diese aber auf Wunsch zur Verfügung gestellt wird.

Transparenz

Zusätzlich zum Verständnis, wann eine Empfehlung oder ein Hinweis angezeigt wird, muss der/die Nutzende auch verstehen, aus welchem Grund sie angezeigt wird. Abhängig von der Komplexität des Empfehlungssystems muss im Designprozess entschieden werden, welche Informationen der/die Nutzende benötigt, um zu verstehen, warum ein Hinweis gegeben wird. Dieser Aspekt steht in Wechselwirkung mit der Verständlichkeit. Werden dem/der Nutzenden zu viele Informationen über den Grund des Hinweises gegeben, erhöht sich die kognitive Belastung, um den Sachverhalt zu verstehen. Konkrete Erklärungen sind jedoch in der Regel abstrakten vorzuziehen.

Zugänglichkeit

Nutzende müssen in der Lage sein zu beurteilen, ob die angezeigten Empfehlungen relevant sind und ob Maßnahmen getroffen werden sollten. In der Regel kann die Zugänglichkeit verbessert werden, indem ein Vergleich zwischen der Empfehlung und Alternativen hergestellt wird. Dieser Aspekt hängt von der Komplexität des Systems ab. Je einfacher ein System konzipiert ist, desto einfacher ist es, die Relevanz zu bewerten.

Vertrauen

Empfehlungssysteme bringen nicht den gewünschten Mehrwert, wenn die Nutzenden nicht davon überzeugt sind, von den Empfehlungen und Hinweisen zu profitieren. Es ist wichtig, Vertrauen in das System aufzubauen. Systeme, die Empfehlungen automatisch umsetzen, erfordern mehr Vertrauen als Systeme, bei denen die Nutzenden die Umsetzung der Empfehlung bestätigen muss. Ebenso schaffen mehrere kleinteilige Empfehlungen mehr Vertrauen als ein System, das den Nutzenden die Bedienung aus der Hand nimmt.

Ablenkung

Der letzte Aspekt ist Ablenkung. Empfehlungen sollten nicht zu früh oder zu oft gegeben werden. Die Nutzenden sollten nicht von ihrer Hauptaufgabe abgelenkt oder unterbrochen werden. Hier müssen die Kosten der Unterbrechung und der versprochene Nutzen gegeneinander abgewogen

werden. So lenken z.B. Hinweise, die Teile der Oberfläche farblich unterlegen, aber ansonsten das Programm nicht einschränken, weniger ab als Hinweise, die den gesamten Bildschirm blockieren.

Diese fünf Aspekte von Murphy und Murphy-Hill [24] sollten bei der Implementierung zusätzlich berücksichtigt werden. Auf den Einfluss dieser Aspekte auf das UI-Design wird in den Kapiteln User Interface und Implementierung des Prototyps näher eingegangen.

4.4 Szenarien

Um die abstrakte Idee aus dem *Vision Statement* zu konkretisieren und die Funktionen aus dem Workshop mit einzubinden, werden im Folgenden Anwendungsszenarien beschrieben. Dabei wird ein Abfolge von Interaktionen mit dem zu entwickelnden Prototypen dokumentiert. Das dient dazu die Software zu verbildlichen. Aber auch soll die Interaktion zwischen Nutzenden und System und der letztendliche Mehrwert durch die Nutzung ersichtlich werden.

Anwendungsszenario 1

In einem Softwareentwicklungsunternehmen soll eine Anwendung in einem agilen Umfeld entwickelt werden. Das Feature wird von einem Entwickler fertig programmiert und der Code an die Qualitätssicherung übergeben. Ein Mitarbeiter testet das Feature und stellt fest, dass es gelegentlich zum Absturz der Software führt. Da es bereits Freitag ist und das Feature spätestens am Montagabend freigegeben werden soll, schreibt er den zuständigen Entwickler direkt über die firmeninterne Kommunikationsplattform an. Er schreibt in das Eingabefeld „Der Scheiß funktioniert überhaupt nicht! Überarbeite das sofort.“. Die Stimmungsanalyse erkennt die Stimmung in der Nachricht als negativ und das Eingabefeld färbt sich rot. Das System zeigt damit an, dass die Nachricht eine negative Stimmung auf den Empfänger übertragen könnte. Klickt der Verfasser der Nachricht nun auf den rot hinterlegten Text, öffnet sich ein Hinweisdialog, der erklärt, was die rötliche Färbung bedeutet. Gleichzeitig wird eine umformulierte Nachricht angezeigt. Diese umformulierte Nachricht behält den Inhalt des Satzes bei, schreibt ihn aber in eine positivere Form um. In diesem Fall lautet die neue Meldung „Das funktioniert so leider nicht richtig. Das muss noch einmal überarbeitet werden.“. Der Autor denkt also noch einmal über seinen geschriebenen Satz nach und erkennt, dass der neu formulierte Vorschlag viel freundlicher klingt als sein Satz. Da er den Entwickler nicht verletzen will, entscheidet er sich, den neuen Satz im Hinweisdialog anzuklicken, um ihn in das Textfeld zu übernehmen. Der Hintergrund färbt sich nun nicht mehr rötlich, sondern grünlich und er schickt die Nachricht an den Entwickler.

Anwendungsszenario 2

Der Projektmanager hat von einem Kunden eine neue Anforderung für ein Feature an das Softwareprodukt in seinem Unternehmen erhalten. Er möchte nun herausfinden, ob und mit welchem Aufwand das neue Feature umgesetzt werden kann. Dabei ist es ihm wichtig zu verstehen, wo die technischen Herausforderungen liegen, die das neue Feature mit sich bringt. Zu diesem Zweck verfasst er in der firmeninternen Kommunikationsplattform in einer Gruppe mit den beteiligten Entwicklern eine Nachricht. Darin erklärt er zunächst, welche Funktionalität das gewünschte Feature haben soll und welche Vorstellungen der Kunde hat. Am Ende der Nachricht formuliert er nun sein Anliegen an die Entwickler: „Können wir das wie gewünscht umsetzen? Wie lange würde das dauern und was ist zu beachten?“. Der Projektleiter schickt die Nachricht ab. Unmittelbar nach dem Absenden bemerkt er einen Tippfehler und schickt sofort das korrigierte Wort hinterher. Die Software erkennt, dass ein Wort korrigiert werden muss und schlägt dem Verfasser in einem Hinweisdialog vor, den Fehler in der letzten Nachricht automatisch zu korrigieren, anstatt eine neue Nachricht nachzuschicken. Der Projektmanager bestätigt und die letzte Nachricht wird korrigiert. Ein Entwickler erhält die Nachricht und liest sie. Da er gerade viel zu tun hat, schreibt er unbedacht nur einen kurzen Satz „Das geht nicht.“. Als er die Nachricht absenden will, zeigt das Kommunikationsprogramm den Hinweis an, dass zwischen der ursprünglichen Nachricht des Projektmanagers und seiner Antwort ein großer Längenunterschied besteht. Nach kurzem Nachdenken erkennt er, dass seine Antwort nicht dem Anliegen des Projektmanagers entspricht. Er beschließt, seine Nachricht nicht abzuschicken und nimmt sich darauf hin die Zeit, seine Meinung über die Schwierigkeiten bei der Implementierung der Funktion besser zu begründen.

4.5 User Interface

Die Benutzeroberfläche des Assistenten soll so gestaltet sein, dass die Nutzenden auf bereits vorhandenes Wissen zurückgreifen können. Es sollen keine neuen Interaktionstechniken erlernt werden müssen. Ebenso soll die Funktionalität des zugrundeliegenden Messaging-Systems nicht beeinträchtigt oder verändert werden. Der Grund dafür ist, dass ein Plattformwechsel mit viel Aufwand verbunden ist. Der Prototyp soll vielmehr als Plugin, also als Erweiterung eines bestehenden Systems, konzipiert werden, das unabhängig von der Kommunikationsplattform funktioniert und eingebettet werden kann. Ebenso soll sich die Benutzeroberfläche über die der Kommunikationsplattform „drüber“ legen. Als Inspiration dienen hier, wie in Kapitel 3 beschrieben, Schreibassistenten wie Grammarly. Dieser kann auch als Browser-Plugin in bestehende Anwendungen integriert werden. Der analysierte Text wird in der externen Anwendung farblich markiert und

anklickbar gemacht. Weitere Hinweise und Empfehlungen werden nach dem Anklicken in einem Tooltip angezeigt beziehungsweise erläutert.

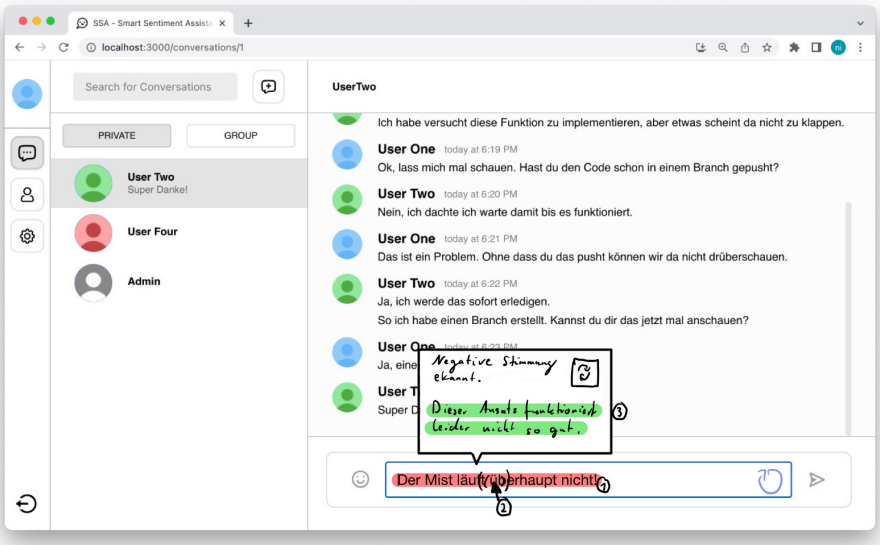


Abbildung 4.2: Skizze der grundlegenden Funktionsweise des Prototypen

In Abbildung 4.2 ist ein Mockup des geplanten Assistenten zu sehen. Als Basis wurde hier der in Kapitel 5 vorgestellte *Chat-Plattform React* verwendet. Dieser wurde bis auf das Ausblenden einiger für diese Arbeit unwichtiger Funktionen in seiner Funktionsweise nicht verändert. Negative Meldungen sollen analog zu Rechtschreibfehlern und schlechtem Stil bei Grammarly rot markiert werden und erst nach einem Klick auf diese Markierung wird ein Tooltip angezeigt. Dieses wiederkehrende Verhalten, das Nutzende bereits aus anderen Anwendungen kennen, erhöht die Verständlichkeit des Empfehlungssystems. Der Grund dafür, dass erst nach dem Klicken auf eine Markierung weitere Informationen angezeigt werden, liegt darin, die Ablenkung durch das System gering zu halten. Nutzende sind nicht gezwungen, weiter mit dem System zu interagieren und kann auch Nachrichten versenden, ohne auf die Ergebnisse der Prüfung zu reagieren. Zusätzliche Verständlichkeit und Zugänglichkeit wird durch die Erklärung im Tooltip erreicht. Würde der umformulierte Satz ohne weitere Erklärung angezeigt, könnte der/die Nutzende nicht direkt verstehen, worin der Unterschied zwischen den Formulierungen (der eigenen und der umformulierten) besteht und warum die Umformulierung vorgenommen wurde.

Statt der manuellen Übernahme der Umformulierung wäre auch ein Ansatz denkbar, den eingegebenen Text automatisch durch die Umformulierung zu ersetzen. Es wird jedoch davon ausgegangen, dass das Vertrauen in das

System dafür nicht hoch genug ist. Ein höheres Vertrauen wird dadurch erreicht, dass die Nutzenden kleinere Einzelempfehlungen erhalten und jedes Mal vor die Wahl gestellt werden, inwieweit sie der Empfehlung folgen möchten.

Kapitel 5

Implementierung

Nachdem in Kapitel 4 das Konzept für den Prototypen entwickelt und die Features und Anforderungen erarbeitet wurden, beschäftigt sich dieses Kapitel mit der Implementierung. Dabei werden die verwendeten Frameworks, Techniken und Libraries vorgestellt und die Architektur des Beispielsystems erläutert. Dazu wird zunächst eine Open-Source-Chat-Plattform beschrieben, die als Basis für den Sentiment-Assistenten dieser Arbeit dient.

5.1 Verwendete Open-Source-Chat-Plattform

Die wichtigste Anforderung an die Chat-Plattform, auf der der Assistent aufbaut, ist zunächst, dass der Quellcode frei verfügbar ist und somit Änderungen am Code vorgenommen werden können. Als Basis wurde die *Chat-Plattform React*[1]. Dieses Projekt bietet für das Ziel der Arbeit mehrere Vorteile. Zunächst handelt es sich um eine Kommunikationsplattform, die in ihren Grundfunktionen bekannten Plattformen wie Microsoft Teams, WhatsApp oder Element ähnelt. Dadurch eignet sich diese Plattform für den in dieser Arbeit zu entwickelnden Assistenten. Nach dem Start der Anwendung muss der/die Nutzende zunächst einen Account anlegen und kann sich dann mit diesem einloggen. Danach können Chats mit einem oder mehreren anderen Nutzenden erstellt werden. Der Chat funktioniert so, wie es die Nutzenden bereits von Whatsapp und ähnlichen Messengern kennen. Es können Nachrichten in einem Textfeld verfasst und versendet werden. Diese werden dann in chronologischer Reihenfolge von oben (älteste Nachricht) nach unten (neueste Nachricht) mit Profilbild, Namen und Zeitstempel angezeigt. Zusätzlich gibt es eine Funktion für Sprachanrufe, die aber im Rahmen dieser Arbeit deaktiviert wird.

Die Software wurde von Anson Foong[2] entwickelt und ist unter CC BY-

[1] https://github.com/stuyy/chat-platform-react | zuletzt aufgerufen am: 09.06.2023
[2] https://github.com/stuyy

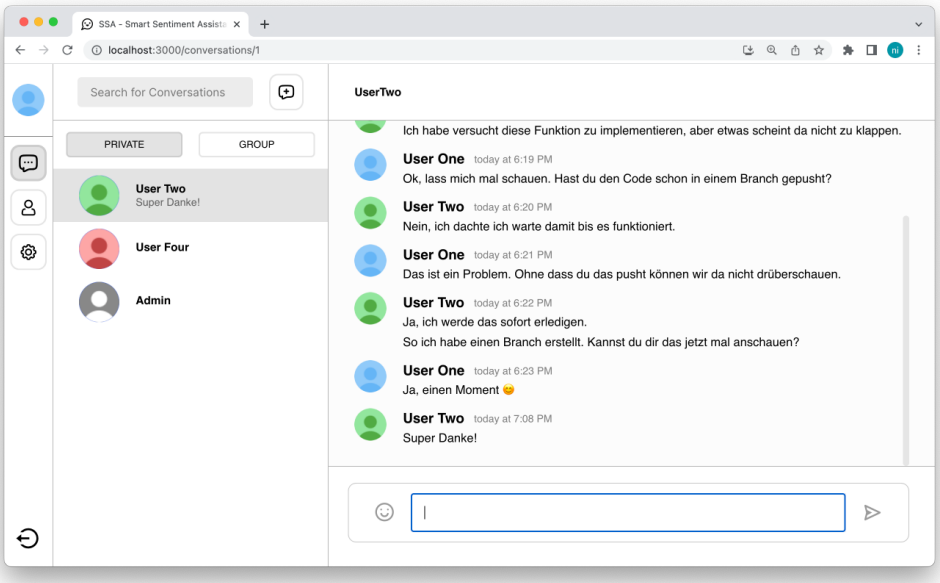


Abbildung 5.1: Hauptansicht des React-Messengers

NC-ND 4.0[3] lizensiert. Das bedeutet, dass der Quellcode unter den folgenden Bedingungen kopiert und weiterverteilt werden darf:

- Es müssen angemessene Urheber- und Rechtsangaben gemacht werden.
- Das Material darf nicht für kommerzielle Zwecke verwendet werden.
- Wenn das Material verändert wird, darf das modifizierte Material nicht veröffentlicht werden.

Die Chat-Plattform wurde, wie der Name schon sagt, mit der Programmbibliothek React von Meta entwickelt. Als Programmiersprache wird TypeScript verwendet. TypeScript basiert auf JavaScript, bietet aber eine statische Typisierung. Dies hat den Vorteil, dass der generierte Code in den meisten Fällen robuster und besser wartbar ist. Auf dieser Basis können UI-Komponenten für den Einsatz auf webbasierten Plattformen, wie zum Beispiel in einem Internetbrowser, entwickelt werden. Dies hat gegenüber einer nativen Windows-, Mac- oder Linux-Anwendung den Vorteil, dass die Anwendung auf allen drei Betriebssystemen gerendert werden kann.

Ein weiterer Vorteil des gewählten Projekts ist die Bereitstellung eines Backend-Systems, das eine lokale Hosting-Möglichkeit bietet. Dieses Backend

[3] https://creativecommons.org/licenses/by-nc-nd/4.0/ | zuletzt aufgerufen am 09.06.2023

wurde mit Node.js realisiert und kann mit einer MySQL-Datenbank betrieben werden. Somit kann die gesamte Plattform in einem geschützten Rahmen getestet und genutzt werden, ohne auf kommerzielle Anbieter zurückgreifen zu müssen. Dieser Messenger eignet sich daher gut als Basis für den Assistenten.

5.2 Softwarearchitektur

Die Logik des Assistenten wurde in Python entwickelt. Für die Wahl von Python gibt es mehrere Gründe. Zum einen bietet Python eine große Auswahl an Bibliotheken, die speziell für die Verarbeitung natürlicher Sprache entwickelt wurden. Außerdem werden in Python viele Machine Learning Tools und Schnittstellen angeboten, die für das Ziel dieser Arbeit hilfreich sind. Außerdem lässt sich Python problemlos mit anderen Technologien und Frameworks integrieren.

So nutzt dieser Prototyp das Flask Web Framework in Python. Es bietet eine Schnittstelle zwischen klassischen Webanwendungen und Python-Anwendungen. So kann auf die Funktionen der Anwendung über Rest-API-Methoden wie *POST* oder *GET* zugegriffen werden. Flask kann mit wenigen Zeilen Code in eine bestehende Python-Anwendung integriert werden.

Abbildung 5.2 zeigt die gesamte Struktur der Software. Man sieht, dass das Chat-Frontend einerseits mit dem Chat-Backend und andererseits mit der Flask-Applikation kommuniziert. Beide Komponenten laufen zusammen mit der Datenbank, die auf MySQL[4] basiert, in einem sogenannten Docker Container. Dabei handelt es sich um eine portable Laufzeitumgebung, die es Entwicklern ermöglicht, mehrere Programme als zusammengefasste und isolierte Einheit auszuführen und zu verwalten. Der Vorteil ist, dass einmal erstellte Programme unabhängig von der Host-Umgebung laufen und keine externen Abhängigkeiten von installierten Bibliotheken oder Konfigurationsdateien haben. Dadurch kann der Entwickler sicherstellen, dass sich die Software auf dem Zielsystem konsistent zur Entwicklungsumgebung verhält [3].

Die Verwendung der Komponente *Huggingface* wird zu einem späteren Zeitpunkt erläutert. Im folgenden Kapitel wird zudem deutlich, welche Teile der Funktionen im Web-Frontend und welche Teile in der Flask-Applikation implementiert wurden.

[4]https://www.mysql.com/de | zuletzt aufgerufen am: 09.06.2023

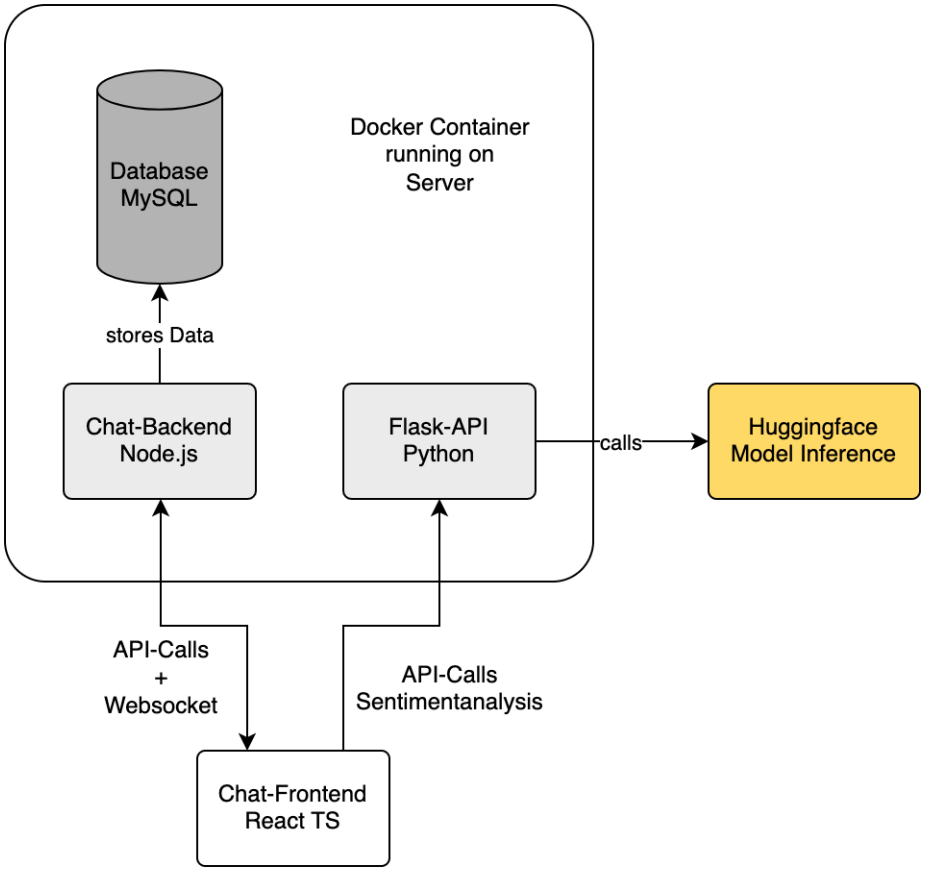


Abbildung 5.2: Übersicht der Softwarearchitektur

5.3 Umsetzungen

In diesem Kapitel werden die in Kapitel 4 erarbeiteten Funktionalitäten vorgestellt. Dabei wird auf die genaue Funktionsweise und die technische Umsetzung eingegangen. Zunächst wird die Sentimentanalyse mit der zugehörigen automatischen Umformulierung als Hauptfokus erläutert. Anschließend wird auf die zusätzlichen Funktionen des Assistenten eingegangen.

5.3.1 Live-Sentimentanalyse

Die Live-Sentimentanalyse soll den Anwendenden bei der Eingabe einer Nachricht in das Textfeld eines Chats anzeigen, wie die Stimmung der Nachricht von einem Sentimentanalysetool bewertet wird. Aufgrund

der Tatsache, dass sachliche Konversationen nicht nur mit explizit positiven Nachrichten geführt werden können, wurde entschieden, dass das Sentimentanalysesystem nur zwischen *negativen* und *nicht negativen* Nachrichten unterscheidet. So wird nicht zwischen *neutralen* und *positiven* Nachrichten unterschieden. Nachdem der/die Nutzende eine Nachricht in das Textfeld im Frontend eingegeben hat, wird nach einer Verzögerung von zwei Sekunden ein Aufruf an die API des Flask-Servers gesendet. In diesem Aufruf wird der Inhalt des Textfeldes gesendet. Der Grund für die Verzögerung ist, dass die Anwendung wartet, bis der/die Nutzende nicht mehr tippt, bevor die Sentimentanalyse gestartet wird. Würde die Analyse hingegen nach jedem neu eingegebenen Zeichen gestartet, würde die Performance darunter leiden. Außerdem wäre es für die Benutzererfahrung nicht sinnvoll, wenn unfertige Nachrichten analysiert würden, da die Sentimentanalyse keine verwertbaren Ergebnisse liefert, wenn ein Satz unvollständig ist oder noch Rechtschreibfehler enthält. Auf der Serverseite werden die einzelnen Sätze der Nachricht zunächst getrennt und dann jeweils mit einem BERT *Transformer* Modell [13] klassifiziert. Die Wahl fiel hierbei wegen den in Kapitel 2 vorgestellten Vorteilen in dem Verstehen von natürlichsprachlichen Text auf ein *Transformer*-basiertem Tool. Das verwendete Modell basiert auf der in Abschnitt 2.3 vorgestellten BERT-Architektur von Google und wurde mit 1,834 Millionen deutschsprachigen Beispieldaten trainiert. Der verwendete Datensatz deckt dabei viele verschiedene Domänen wie Twitter, Facebook aber auch App Reviews ab. Das Modell wurde wie oben beschrieben speziell für Dialogsysteme entwickelt. *SentiStrength* wurde mit der Erweiterung *SentiStrength-SE* zwar auf die Domäne Software-Engineering angepasst, legt den Fokus allerdings nicht auf direkter Kommunikation in einem Kommunikationstool sondern es wurde auf Basis von Issues in Jira konzipiert [16]. Aus diesem Grund und wegen der deutschen Sprache wurde dieses Modell für die Implementierung im Prototyp ausgewählt.

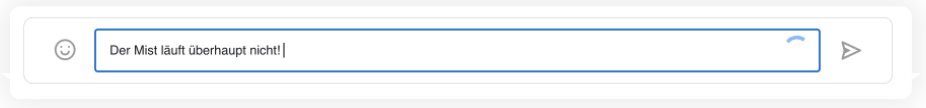


Abbildung 5.3: Ladeindikator auf der rechten Seite des Textfeldes

Bei der Verarbeitung der Daten im Python-Backend wird im Eingabefeld des Chats, wie in Abbildung 5.8 zu sehen, ein Ladekreis angezeigt. Dieser informiert Nutzende darüber, dass die Eingabe derzeit verarbeitet wird. Dies soll die Nutzenden dazu veranlassen, auf das Ergebnis der Analyse

zu warten, bevor sie die Nachricht abschicken. Auch fördert das die Transparenz in das System, da Nutzende zu jedem Zeitpunkt wissen, ob gerade eine Analyse vorgenommen wird oder nicht. Die Analyse dauert circa 1-2 Sekunden.

Abbildung 5.4: Markierung der negativen Stimmung im Textfeld des Prototypen

Nachdem die Analyse abgeschlossen ist, gibt die Python-Anwendung das Ergebnis für jeden Satz an das Frontend zurück. Ist das Ergebnis für einen Satz *Negativ*, wird dieser Text rot markiert. Dies erhöht die Verständlichkeit des Systems, da Nutzende ähnliche Markierungen bereits von der Rechtschreibprüfung in Microsoft Word oder der Stilprüfung in Grammarly kennen. Die Farbe Rot wurde als Signalfarbe gewählt, da sie in der Regel auf etwas Negatives hinweist.

Ein rot hinterlegter Satz ist anklickbar. Dies wird durch eine Veränderung des Cursors und eine Veränderung der Hintergrundfarbe beim Überfahren mit der Maus angezeigt. An dieser Stelle wird bewusst nicht der umformulierte Satz oder eine andere Handlungsempfehlung angezeigt, um den Nutzenden nicht unnötig zu unterbrechen oder abzulenken. Wenn das BERT-Modell für keinen Satz eine *negative* Klassifizierung zurück, wird das gesamte Textfeld in einem blassen Grünton dargestellt. Dies steht im Kontrast zur roten Farbe und greift die Metapher der Signalfarbe auf. Dies ist ein Zeichen für den/der Benutzenden, dass das System keine *negative* Stimmung erkannt hat und daher keine Umformulierung generiert.

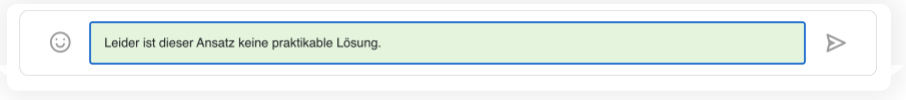


Abbildung 5.5: Markierung der positiven Stimmung im Textfeld des Prototypen

5.3.2 Umformulierung von negativen Nachrichten

Wenn ein negativer Satz erkannt wird, soll das System einen Vorschlag für die Umformulierung des Satzes in einen positiveren Satz machen. Dabei soll explizit kein rein positiver Satz gebildet werden, um die semantische Bedeutung des Satzes nicht zu stark zu verändern. Wenn das Sentiment Classification Model einen Satz als *negativ* klassifiziert hat, wird der Satz vom LLaMa (Large Language Model Meta AI) umformuliert. Da diese Art von Sprachmodellen jedoch für den Einsatz auf GPU-basierter Hardware konzipiert ist, können sie auf den meisten Verbrauchercomputern nicht ausgeführt werden. Abhilfe schaffen hier die HuggingFace Spaces[5]. HuggingFace ist eine Webplattform, die es im Bereich des maschinellen Lernens Nutzern ermöglicht, verschiedene vortrainierte Modelle, Datensätze und Bibliotheken gemeinsam zu nutzen. Darüber hinaus bietet die Plattform auch die Möglichkeit eigene Machine Learning Python basierte Anwendungen in sogenannten Spaces hochzuladen und zu betreiben. Diese Anwendungen können, müssen aber nicht öffentlich zugänglich sein. Im Rahmen dieser Arbeit wird ein privater Space verwendet um das LLaMa Sprachmodell zu betreiben. Dieser ist dann ähnlich wie die Flask-Anwendung über die Rest-API erreichbar. Da das Modell nur auf Englisch trainiert wurde, muss sowohl die Eingabe als auch die Ausgabe des Modells aus dem Deutschen bzw. ins Deutsche übersetzt werden.

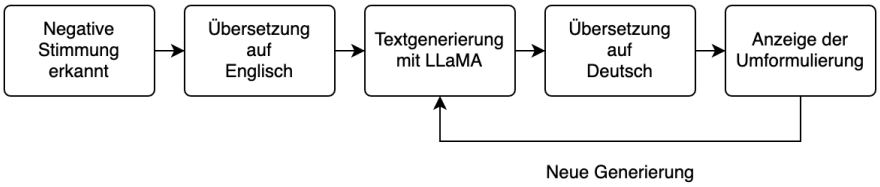


Abbildung 5.6: Verarbeitungsschritte der Umformulierung

Abbildung 5.6 zeigt die Verarbeitungsschritte. Da das Modell in englischer Sprache deutlich bessere Ergebnisse liefert, wurde ein Ansatz gewählt, bei dem die Ein- und Ausgabe in englischer Sprache erfolgt. Federmann et al. [11] gezeigt, dass mit Hilfe von Übersetzungstools geeignete Paraphrasen generiert werden können, indem Sätze aus der Ausgangssprache in andere Sprachen übersetzt und anschließend wieder in die Ausgangssprache zurückübersetzt werden. Da der Ausgangssatz ohnehin umformuliert wird, also eine Paraphrase entsteht, muss nicht sichergestellt werden, dass die Übersetzungen die wortwörtliche Bedeutung perfekt abbilden. Dennoch haben Federmann et al. gezeigt, dass Übersetzungen aus verwandten Sprachen, wie

[5]https://huggingface.co/spaces | zuletzt aufgerufen am: 09.06.2023

Englisch und Deutsch, im Vergleich zu Übersetzungen aus fremden Sprachen, wie Englisch und Mandarin, zwar weniger diverse, aber dafür genauere Paraphrasen bilden.

Der neu generierte Satz wird im Web-Frontend angezeigt, wenn der/die Anwendende auf einen rot hinterlegten negativen Satz klickt. Dazu wird ein Tooltip verwendet, der an der Stelle angezeigt wird, an der der/die Benutzende geklickt hat. Da die Verarbeitung des Satzes etwa 5 bis 10 Sekunden dauert, wird dem/der Benutzende ein Fortschrittsbalken angezeigt, wenn er den Satz zuvor angeklickt hat.

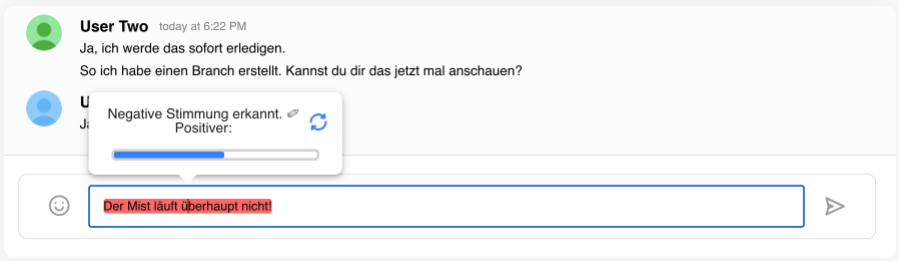


Abbildung 5.7: Fortschrittsbalken bei der Umformulierung der negativen Nachricht

Ist die Generierung des neuen Satzes abgeschlossen, wird dieser im Tooltip angezeigt und die Nutzenden haben die Möglichkeit, den eingegebenen Satz durch den neu generierten Satz zu ersetzen. Mit dem neu generierten Satz wird nun keine Sentiment-Analyse durchgeführt, da es zu einem schlechten Nutzungserlebnis führen würde, wenn der neue Satz wieder als negativ bewertet würde. Daher wird davon ausgegangen, dass der neu generierte Satz eine positivere Variante des alten Satzes ist.

Neben der Möglichkeit, den neuen Satz in das Textfeld zu übernehmen, wird dem Nutzenden auch die Möglichkeit gegeben, durch einen Klick auf das blaue Neuladesymbol eine neue Formulierung generieren zu lassen. Dies soll die Freiheit des Nutzenden erhöhen und das Gefühl verringern, eine Formulierung vorgegeben zu bekommen. Die Formulierungen besitzen eine ausreichend große Varianz, sodass beim Testen der Software kein Fall aufgetreten ist, bei dem zwei Formulierungen hintereinander identisch waren.

5.3.3 Anzeige von Antwortvorschlägen

Für die Generierung von Antwortvorschlägen gibt es verschiedene Ansätze. Grundsätzlich funktionieren Antwortvorschläge ähnlich wie Chatbots. Der Unterschied besteht lediglich darin, dass bei Antwortvorschlägen meist mehr als eine Alternative generiert werden muss, während Chatbots meist auf

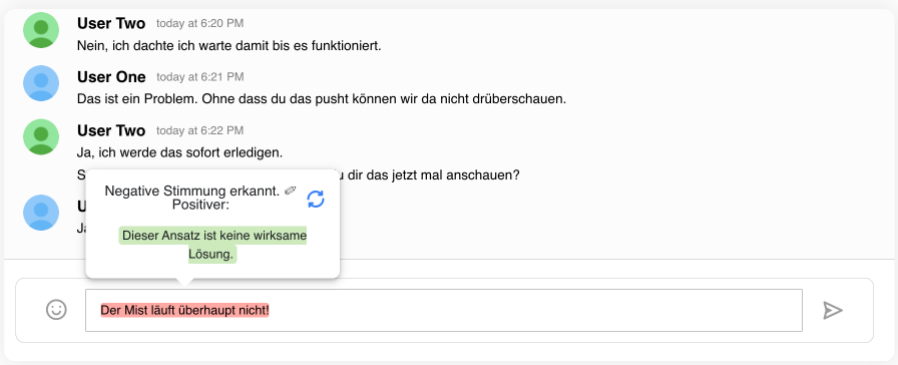


Abbildung 5.8: Anzeige der positiver formulierten Nachricht

jede Nachricht nur eine Nachricht zurückgeben. Ansonsten wird in beiden Fällen auf Basis eines Textes ein oder mehrere passende Antworttexte zurückgegeben. Die Ansätze bestehen zum einen aus regelbasierten Ansätzen. Dabei wird anhand von vordefinierten Regeln eine vordefinierte Antwort aus einer Datenbank ausgewählt. Diese Regeln können je nach Implementierung einfach oder komplex sein. Damit ein solches System zuverlässig Antworten generiert, müssen sehr viele Regeln und Antwortmöglichkeiten manuell eingegeben und verschiedenste Fälle berücksichtigt werden. Generative Chatbots gehen anders vor. Die Antworten werden, ähnlich wie bei der Implementierung von Umformulierungen, von einer generativen künstlichen Intelligenz aus den Ausgangsnachrichten generiert. Da jedoch auch dieser Ansatz ressourcenintensiv ist und der Fokus dieser Arbeit auf der Umformulierung von Nachrichten liegt, wurde ein *Retrieval-based* Ansatz gewählt. Bei *Retrieval-based* Modellen werden Heuristiken verwendet, um aus einer Sammlung von vordefinierten Antworten diejenigen auszuwählen, die am besten zum Kontext passen. In dieser Arbeit wird der Eingabetext in eine Vektorrepräsentation umgewandelt. Anschließend wird die Kosinusähnlichkeit zwischen der Eingabe und vordefinierten Mustern berechnet und die am besten passenden Antwortmöglichkeiten zurückgegeben. Bei der Erstellung der Sammlung wurde besonderes Augenmerk auf Muster gelegt, die auf eine Frage hindeuten, da, wie in Kapitel 4 beschrieben, das Ziel der automatischen Antwortvorschläge ist, die Antwortzeit der Nutzenden auf Fragen zu verkürzen. Eine vollständige Liste der Muster und Antwortmöglichkeiten ist im Anhang B zu finden. Zu beachten ist, dass die Antworten teilweise nur aus Satzanfängen bestehen, wenn zum Beispiel eine Erläuterung eines Sachverhalts erwartet wird. Es würde die Komplexität des Systems und den Rahmen dieser Arbeit sprengen, auf den Inhalt der Ausgangsnachricht näher einzugehen.

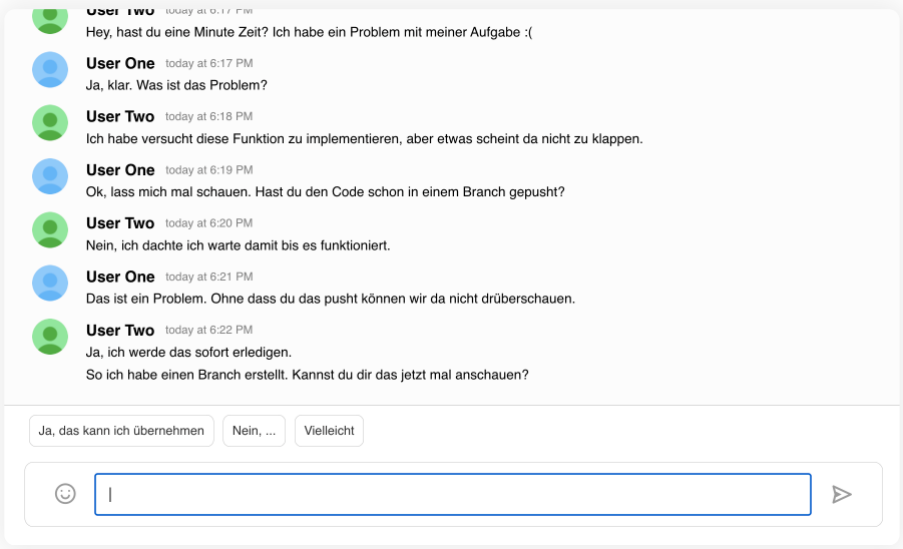


Abbildung 5.9: Anzeige der automatisch vorgeschlagenen Antwortmöglichkeiten

In der Benutzeroberfläche werden die Vorschläge, ähnlich wie bei vergleichbaren Messengern, beim Eintreffen einer Nachricht über dem Textfeld angezeigt. Die Berechnung dauert nur wenige Millisekunden. Klickt der/die Benutzende auf einen Vorschlag, wird dieser in das Textfeld übernommen und das Textfeld fokussiert.

5.3.4 Rechtschreibkorrektur

Die Rechtschreibkorrektur erfolgt auf der technischen Seite von Hunspell[6]. Dabei handelt es sich um ein Rechtschreibprüfungstool für Sprachen mit komplexen Wortzusammensetzungen. Hunspell zerlegt Wörter in kleinere Teile und analysiert diese separat. So kann ein Wort auch dann als richtig erkannt werden, wenn sowohl der Wortstamm als auch die Endung korrekt sind. Alle Wörter mit allen Formen zu speichern, würde zu einem zu großen Wörterbuch und zu langen Berechnungszeiten führen. Das Programm wurde ursprünglich für die ungarische Sprache entwickelt. Es basiert auf MySpell, das für verschiedene Open-Source-Programme wie OpenOffice entwickelt wurde, kann aber im Gegensatz zu diesem UTF-8-kodierte Wörtersammlungen verwenden. UTF-8 ist die am weitesten verbreitete Zeichenkodierung in der Informatik. Heute wird Hunspell zum Beispiel von Mozilla Firefox oder Google Chrome verwendet [15].

[6] http://hunspell.sourceforge.net/ | zuletzt aufgerufen am: 09.06.2023

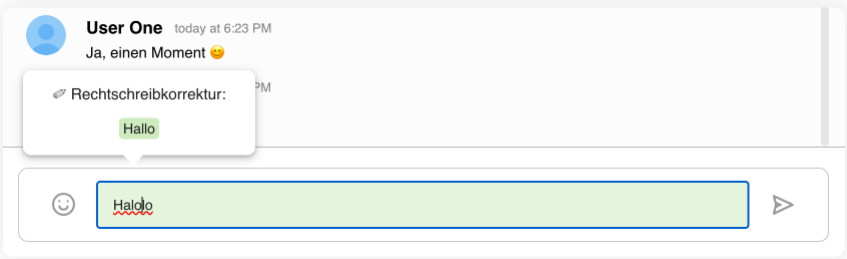


Abbildung 5.10: Anzeige der Rechtschreibhilfe

Um eine größtmögliche Einheitlichkeit in der Benutzeroberfläche zu erreichen, wird der Vorschlag der Rechtschreibprüfung ähnlich wie die Umformulierung angezeigt. Der Unterschied besteht lediglich in der Markierung der Rechtschreibfehler und der Erklärung im Tooltip. Als Markierung wurde hier eine rote Unterkringelung gewählt, da fast alle Systeme, die Rechtschreibfehler anzeigen, eine ähnliche Markierung vornehmen. Dies hat den Vorteil, dass Nutzende die Metapher bereits kennen und ihr Wissen über die Bedeutung direkt anwenden können.

5.3.5 Weitere Funktionen

Zusätzlich zu den bereits vorgestellten Funktionen, wurden weitere Metriken für negative Stimmung untersucht. Schroth [33] hat in seiner Arbeit neben dem Inhalt weitere statistische Metriken genannt, an denen sich die Stimmung ablesen lässt. Die Metriken, die in diese Arbeit aufgenommen wurden, sind die übermäßige Verwendung von Satzzeichen und Großbuchstaben in der Kommunikation. Zum Beispiel ist der Satz „Bitte erledige das jetzt.“ weniger negativ als „Bitte erledige das JETZT.“. Dieser Eindruck kann durch eine Abfrage des oben vorgestellten BERT-Modells zur Sentimentanalyse bestätigt werden. Während der Satz ohne Großschreibung mit 46 Prozent als neutral eingestuft wird, wird der Satz mit dem großgeschriebenen „JETZT“ mit 50 Prozent als negativ bewertet.

Dies gilt analog für die Verwendung von Satzzeichen. „Wie funktioniert die API-Schnittstelle?“ (85 Prozent - Neutral) ist positiver als „Wie funktioniert die API-Schnittstelle?!?“ (47 Prozent - Negativ). Zusätzlich zur Sentimentanalyse wurde in die API-Abfrage an das Flask-Backend eine Funktion eingebaut, die aufeinanderfolgende Großbuchstaben und Satzzeichen erkennt. Diese werden analog zur Sentimentanalyse ebenfalls farblich hervorgehoben. Die Nutzenden haben zudem die Möglichkeit, den bereitgestellten Vorschlag zu übernehmen. Dabei werden Zeichenketten gekürzt und großgeschriebene Wörter in ihrer korrekten Groß- und Kleinschreibung bereitgestellt.

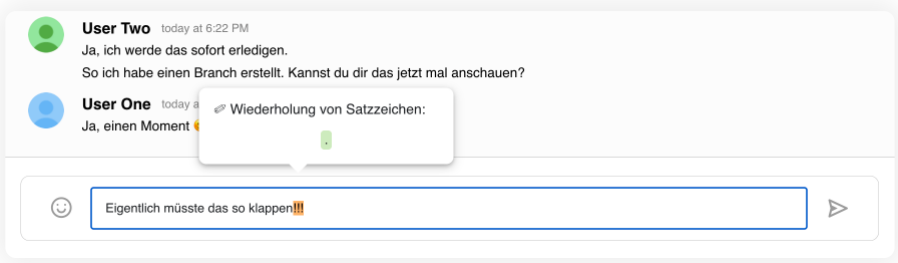


Abbildung 5.11: Hinweis auf mehrere aufeinanderfolgende Satzzeichen

Bei der Erkennung von Großbuchstaben gibt es im Gegensatz zu den Satzzeichen eine Besonderheit in der Umsetzung. Hier wurden mit Hilfe der *Named Entity Recognition* Eigennamen in einem Satz identifiziert. So kann es gerade im Entwicklungskontext vorkommen, dass Firmen- oder Technologieabkürzungen mit Großbuchstaben dargestellt werden. Beispiele dafür sind „SQL‘, „API‘ oder „ASUS‘. Diese Ausdrücke sollten, wenn möglich, bei der Analyse ignoriert werden.

Zuletzt werden zwei Dialoge vorgestellt, die von der Software geöffnet werden können, um zusätzliche Faktoren zu verhindern, die zu einer Übertragung von schlechter Stimmung führen können. Zum einen handelt es sich um einen Hinweis, dass die vorherige Nachricht bearbeitet werden kann. Dieser Hinweis erscheint, wenn eine Nachricht gesendet wird, die nur aus einem Wort besteht und dieses Wort eine Levenshtein-Ähnlichkeit von größer oder gleich 0,5 zu einem Wort der vorherigen Nachricht aufweist [25]. Insbesondere wenn das geschriebene Wort mit einem „*“ beginnt. Dies soll das Phänomen verhindern, dass Nutzende ihre Rechtschreibfehler mit einer schnellen zweiten Nachricht „hinterher korrigieren“. Das direkte Editieren des ursprünglichen Textes erhöht die Lesbarkeit des Chats und reduziert die Anzahl der Nachrichten.

5.4 Nicht verwendete Ansätze zur Umsetzung

Neben dem oben beschriebenen Verfahren zur Generierung positiver Umformulierungen mit einem Large Language Model wurden im Rahmen dieser Arbeit weitere Verfahren und technische Lösungsansätze implementiert, aber wieder verworfen. So wurde versucht, die Umformulierung mit klassischen Verfahren zu realisieren, die nicht auf künstlicher Intelligenz basieren. So wurde ein Ansatz implementiert, bei dem zusätzlich über Sentimentanalyse mit einem BERT-Modell *SentiStrength* verwendet wurde, um die negativen Wörter aus dem Satz zu identifizieren. Ziel war es, die identifizierten negati-

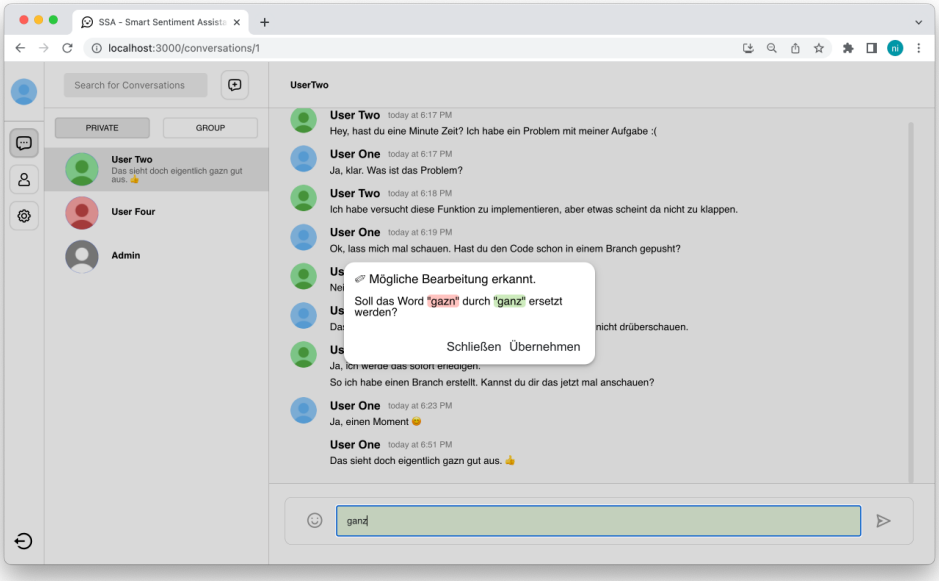


Abbildung 5.12: Hinweis zur Editierung einer Nachricht

ven Wörter im umformulierten Satz durch positivere Synonyme zu ersetzen, um die Gesamtstimmung des Satzes zu verbessern. Um positivere Synonyme zu finden, wurde das SentiWordNet [5] verwendet. Dieses SentiWordNet basiert auf WordNet, einer lexikalischen Datenbank, die englischsprachige Wörter in Gruppen von Synonymen, so genannte *Synsets*, einteilt. Dabei werden semantische Bedeutungen in unterschiedlichen Kontexten berücksichtigt, so dass dieselben Wörter in ihren unterschiedlichen Bedeutungen und Kontexten mehrfach im Wortnetz vorkommen. In der Regel sind alle Wörter über verschiedene Hierarchiestufen miteinander verknüpft. SentiWordNet erweitert das WordNet dahingehend, dass jedem Wort eine Positivität, Negativität und Objektivität zugeordnet wird. Jedes Auftreten eines Wortes hat eine Punktzahl für alle drei Werte, die in Summe 1 ergeben.

Um das richtige Auftreten eines Wortes im SentiWordNet zu finden, beispielsweise wenn ein Wort sowohl Verb als auch Substantiv sein kann, muss zunächst ein sogenanntes *Part-of-Speech-Tagging* auf dem Ausgangssatz durchgeführt werden. Dabei wird jedem Wort seine Rolle im Satz zugewiesen, so dass zwischen Substantiven, Adjektiven, Adverbien etc. unterschieden werden kann. Auf diese Weise können die semantisch passenden Synsets gefunden werden. Nun kann das Synonym im jeweiligen Synset mit dem geringsten Negativitätswert identifiziert und das ursprüngliche Wort ersetzt werden.

Nach ersten Tests hat sich diese Lösung jedoch als unzureichend erwiesen. Der Grund dafür ist, dass sich die Sätze in ihrer Struktur und Bedeutung zu wenig verändern. So wird beispielsweise der Satz „Die Präsentation war langweilig und einfallslos.“ durch die Ersetzung der Wörter „öde“ und „einfallslos“ zu „Die Präsentation war langweilig und uninspirierend.“. Dieses Beispiel zeigt, dass der Unterschied in der Bewertung der Stimmung relativ gering ist. Außerdem würden bestimmte Wörter, die ein Synonym mit weniger negativer Bewertung haben, dauerhaft unterdrückt. So würde das System „öde“ immer in „langweilig“ umwandeln und damit „öde“ aus dem Sprachgebrauch verbannen. Es kann auch vorkommen, dass Wörter durch andere ersetzt werden, die sprachlich nicht zum Rest des Satzes passen, weil sie zu wissenschaftlich und objektiv sind. So wird zum Beispiel „Du bist immer zu spät zum Meeting!“ umformuliert zu „Du bist kontinuierlich zu spät im Meeting!“. Hier wirkt das Wort „kontinuierlich“ nicht passend.

Kapitel 6

Evaluation

Im folgenden Kapitel wird der entwickelte Prototyp im Rahmen einer qualitativen Studie untersucht. Dazu wird zunächst das Ziel der Studie definiert und der Aufbau der Nutzerstudie erläutert. Des Weiteren wird auf die Datenerhebung und -auswertung während der Studie eingegangen.

6.1 Ziel

Um das Ziel und die zu beantwortenden Fragen für die Evaluation zu formulieren, wird das *Goal-Question-Metric-Modell* angewendet. Dabei wird zunächst ein Hauptziel unter Berücksichtigung der fünf Aspekte eines GQM-Ziels definiert. Diese sind das Objekt der Messung, der Zweck, der Qualitätsfokus, der Blickwinkel und der Kontext. Ausgehend von diesem Ziel werden im Folgenden Leitfragen für die Studie formuliert und die entsprechenden Metriken beschrieben, die zur Beantwortung der Leitfragen notwendig sind. [40]

Ziel	Analysiere	die gewinnbringende Einsetzbarkeit des Schreibassistenten
	zum Zwecke	der Vermeidung der Weitergabe negativer Stimmung
	in Bezug auf	schriftlichen Kommunikation
	vom Blickwinkel	eines Entwicklungsteams
	im Kontext	eines Softwareprojektes.

Ausgehend von diesem Ziel ergeben sich die folgenden Leitfragen inklusive der beschriebenen Metriken. Diese Metriken beschreiben die Messwerte, auf deren Basis eine Leitfrage beantwortet werden kann. Dies können tatsächlich messbare Metriken sein, aber auch subjektive Bewertungen der Studienteilnehmenden. Um die oben erwähnte gewinnbringende Einsetzbarkeit zu evaluieren, soll die Studie vor allem

zeigen, ob und wie häufig die Anwendenden die Funktionen der Software nutzen und sie bewerten.

Abbildung 6.1: Auflistung der Leitfragen und der dazugehörigen Metriken

6.2 Planung

Zur Beantwortung dieser Fragen wird im Folgenden eine Gruppen-Laborstudie entwickelt. Die Evaluation der Software, zum Beispiel im Rahmen von einzelnen Think-Aloud-Studien, wird als ungeeignet angesehen. Der Grund dafür ist, dass die Studie die Auswirkungen der Software auf die Kommunikation zeigen soll. Eine solche Kommunikation ist in Studien, die nur von einzelnen Personen gleichzeitig durchgeführt werden, schwierig zu erzeugen. Daher wird ein Ansatz gewählt, bei dem mehrere Personen gleichzeitig ein kleines Softwareprojekt lösen und dabei den entwickelten Prototypen als Kommunikationsplattform nutzen.

Um den Prototypen in einem möglichst realistischen Umfeld zu evaluieren, wird nun ein Szenario vorgestellt, in dessen Kontext der Prototyp eingesetzt werden kann. Das Szenario soll dabei die Aspekte der Perspektive eines Entwicklungsteams und den Kontext eines Softwareprojektes möglichst realitätsnah abbilden.

6.3 Setting

Die Laborstudie findet in Präsenz in einem für die Studie zur Verfügung gestellten Computerraum statt. Für jeden Studienteilnehmer wird ein Arbeitsplatz mit einem PC zur Verfügung gestellt. Die Probanden

werden in zwei verschiedene Gruppen eingeteilt, die unterschiedliche Rollen in einem Softwareprojekt abbilden. Auf der einen Seite sind dies die Entwickelnden, die die Aufgabe bekommen, kurze Aufgaben programmatisch umzusetzen, und auf der anderen Seite die anweisungsgebende Person, die nicht entwickeln, sondern die umzusetzenden Aufgaben an die Entwickelnden kommunizieren.

6.3.1 Aufgabenstellung

Im ersten Schritt, der Entwicklungsphase ergeben sich folgende Rollenanweisungen. Die Programmieraufgabenstellungen sind im Anhang Abschnitt C.3 zu sehen.

Anweisungsgebende(r): Erhält verschiedene Aufgaben. Er/sie erhält nur die Beispielinputs und die dazugehörigen Outputs. Es muss aus den Inputs und Outputs herausgefunden werden, was die tatsächliche Aufgabenstellung ist bzw. welches Muster dahinter steckt. Er/Sie beschreibt den Programmierenden, was die jeweiligen Funktionen leisten müssen. Weder die Beispiel-Eingaben und -Ausgaben dürfen weitergegeben werden, noch dürfen sich andere Beispiele ausgedacht und weitergegeben werden.

Programmierende(r): Muss aus den Anweisungen des/der Anweisungsgebenden Funktionen entwickeln, die die Aufgaben lösen. Er/sie darf in dieser Phase keine Fragen an die Anweisungsgebenden stellen. Es darf sich jedoch mit den anderen Programmierenden über die Programmiersprache und die Syntax ausgetauscht werden.

Im zweiten Teil der Bearbeitung wird der Programmcode getestet und die entstandenen Fehler sollen von den Akteuren aufgelöst werden.

Programmierende(r): Verkettet die implementierten Funktionen (die Aufgaben sind so gestellt, dass der Output einer Funktion immer der Input der nächsten Funktion sein kann). Die Person bekommt vom Spielleiter eine Gesamteingabe und lässt diese durch seine Funktionen laufen. Die Aufgaben sind so schwierig oder fehleranfällig, dass das Ergebnis wahrscheinlich falsch ist.

Programmierende und Anweisungsgebende: Müssen versuchen zurückzuverfolgen, wo die Fehler entstanden ist. Jetzt kann zwischen den Teilnehmenden Beispiel-Inputs und -Outputs ausgetauscht werden. Fehler können mehrere Ursachen haben: War die Anweisung des/der Programmierenden korrekt oder präzise genug? Hat der/die Programmierende einen unbeabsichtigten Fehler eingebaut? Gab es ein Missverständnis zwischen den Beteiligten?

6.3.2 Quellen für negative Stimmung und Probleme

Ziel ist es, dass im ersten Schritt eine Software entsteht, von der beide Rollen denken, dass sie gut funktioniert und dass die Aufgabe erfolgreich abgeschlossen wird. Allerdings bedingen eine Reihe von Faktoren, dass die von den Programmierenden erstellte Software Fehler enthält, was zu Spannungen zwischen den Teilnehmern führen kann.

Zeitmangel: Die Menge und der Umfang der Aufgaben sind so gewählt, dass sie in der vorgegebenen Zeit nur sehr schwer zu bewältigen sind. Die Anweisungsgebenden sind auch dafür verantwortlich, die Zeit im Auge zu behalten und sich von Zeit zu Zeit über den Entwicklungsstand zu informieren.

Abhängigkeit der Aufgaben: Die Aufgaben sind so gestellt, dass sie von den Programmierenden einzeln gelöst werden können. Bei der Überprüfung der Korrektheit der Lösungen werden jedoch alle Einzelaufgaben in ihrer Gesamtheit betrachtet. Die Aufgaben/Funktionen sind so aufgebaut, dass jeweils die Ausgabe einer Funktion die Eingabe der nächsten darstellt. Dadurch können sich Fehler leicht fortführen und es ist nicht klar, wo der Fehler seinen Ursprung nimmt.

Unklarheiten über die Programmiersprache: Die Teilnehmen wurden im Vorfeld nicht über die zu benutzende Programmiersprache in Kenntnis gesetzt, sodass Probleme mit der Programmiersprache, zum Beispiel der Syntax, auftreten können. Da sie nur Zugriff auf die zur Verfügung gestellten Tools haben, müssen sie diese Probleme im Austausch mit den anderen Teilnehmenden lösen.

6.4 Datenerhebung

Um die Leitfragen mit den Ergebnissen der Studie beantworten zu können, müssen Metriken, die nicht über einen Fragebogen erhoben werden können, von der Software protokolliert werden. Zu diesem Zweck zeichnet der Prototyp alle Interaktionen mit der Software auf. Dazu gehören unter

anderem die Anzahl der geschriebenen Nachrichten und die Ergebnisse der Stimmungsanalyse. Insbesondere bei negativen Stimmungen wird zusätzlich aufgezeichnet, wie oft der rot hinterlegte Satz angeklickt wird, um die Umformulierung zu sehen. Außerdem wird erfasst, ob der umformulierte Satz übernommen oder der Tooltip wieder ausgeblendet wird. Alle diese Daten werden zur Laufzeit des Frontends mit einem Zeitstempel in eine CSV-Datei geschrieben. Dabei wird neben der eigentlichen Aktion auch der jeweilige Inhalt gespeichert. Beim Ausloggen des Benutzers wird diese CSV-Datei lokal auf den benutzten Rechner heruntergeladen. Diese Daten werden also nur auf der Seite des/der Nutzenden verarbeitet und gespeichert und nicht über das Internet versendet.

Zusätzlich zu diesem Protokoll der Interaktionen wird der Bildschirm der Testpersonen mit der freien Aufzeichnungssoftware OBS Studio[1] aufgezeichnet. Dies dient vor allem dazu, den gedanklichen Weg zu den einzelnen Interaktionen besser nachvollziehen zu können. Beispielsweise lässt sich so besser erkennen, ob Probanden zögern, etwas anzuklicken, oder wie lange sie auf die Generierung von Umformulierungen warten.

Neben den messbaren Metriken beinhalten die Leitfragen jeweils auch die subjektiven Bewertungen der Probanden. Um diese persönlichen Bewertungen des Prototyps zu erheben, wird ein Fragebogen mit der freien Online-Umfrage-Applikation LimeSurvey[2] erstellt. Dabei beantwortet der Proband zum einen Fragen zur Demographie und den Erfahrungen mit Messenger-Diensten und Schreibassistenten im beruflichen Kontext. Zum anderen werden Fragen zur selbst wahrgenommenen Nutzungshäufigkeit und zur Bewertung der Funktionen beantwortet. Dabei werden die Fragen auch so gestellt, dass der Proband offen Anmerkungen und Verbesserungsvorschläge machen kann. Schließlich soll die allgemeine Gebrauchstauglichkeit der Software beurteilt werden. Dazu wird die sogenannte System Usability Scale (SUS) [22] verwendet. Ziel dieser Skala ist es, anhand von zehn Fragen die Gebrauchstauglichkeit von Systemen technologie- und kontextunabhängig zu bewerten. Dabei quantifiziert das von John Brooke entwickelte System mit Hilfe der Likert-Skala die Antworten auf die Fragen jeweils auf einer Skala von 0 bis 10, so dass sich eine maximale Punktzahl von 100 (perfekte Gebrauchstauglichkeit) ergibt. Diese und alle anderen Fragen des Fragebogens finden sich in Anhang C.

6.5 Durchführung der Studie

Die vorgestellte Studie wurde am frühen Abend eines Wochentages durchgeführt. Der Grund dafür ist, dass alle Probanden berufstätig und somit

[1] https://obsproject.com/de | zuletzt aufgerufen am: 09.06.2023
[2] https://www.limesurvey.org/de/ | zuletzt aufgerufen am: 09.06.2023

tagsüber beschäftigt sind.

Die Akquise der Probanden erfolgte im persönlichen Umfeld des Autors. Bei der Akquise wurde insbesondere darauf geachtet, dass die Teilnehmenden einen Bezug zu den Themen Informatik und Software Engineering haben. Dieser Bezug basiert optimalerweise auf praktischen Erfahrungen im Beruf oder Studium. Denn der Prototyp soll in einem möglichst realistischen Szenario eingesetzt werden. Darüber hinaus soll die Aufgabe, die einen starken Bezug zu den Aufgaben von Entwicklungsteams hat, lösbar sein, ohne dass zum Beispiel Grundlagen in der Programmierung fehlen. Inwieweit dies gelungen ist, wird in Kapitel 7 dargestellt.

Nach der Begrüßung und der Erläuterung der Aufgabenstellung und der Rollen des Experiments (vgl. Unterabschnitt 6.3.1) teilten sich die Teilnehmenden unter Berücksichtigung ihrer Programmierkenntnisse in der Programmiersprache Java auf die Rollen auf. Nach der Rollenverteilung wurden die Ziele und Aufgaben der einzelnen Rollen noch einmal genauer erläutert und anhand von Beispielen veranschaulicht. Besonderer Wert wurde bei der Erklärung darauf gelegt, dass die Aufgaben nur mit ausreichender Kommunikation über den Prototypen zwischen den Teilnehmern gelöst werden können. Nach weiteren Fragen und dem Hinweis, dass jederzeit Rückfragen zu den Aufgaben gestellt werden können, wurde die Studie gestartet und die Probanden begannen mit der Nutzung des Prototyps.

Während sich zu Beginn der Studie die Rechenzeit für die Stimmungsanalyse und die Umformulierungen noch in dem in Kapitel 5 beschriebenen Rahmen (1-2 Sekunden) bewegte, fiel nach etwa zehn Minuten auf, dass sich diese Rechenzeit deutlich erhöhte. Der Grund dafür war, dass das Backend mit der Verarbeitung der Anfragen von sechs Clients nicht nachkam. Eine genauere Analyse dieses Problems findet sich in Kapitel 8. Dadurch konnten die Funktionen des Prototyps nicht mehr in der vorgesehenen Weise genutzt werden. Um die Programmieraufgaben in der vorgegebenen Zeit zu lösen, warteten die Testpersonen bei der Kommunikation nicht mehr auf die Antwort des Backends. Sie verschickten also die meisten Nachrichten, ohne auf die Ergebnisse der Sentimentanalyse zu warten.

Im zweiten Teil der Studie, dem explorativen Testen der Software, sollten die Probanden die Funktionsweise der Software unabhängig von der Programmieraufgabe untersuchen. Dazu wurden ihnen negative Beispielsätze vorgelegt, wie sie in einem Softwareprojekt vorkommen können. Die Probanden hatten die Aufgabe, entweder diese Sätze zu übernehmen oder sich selbst vergleichbare Sätze auszudenken und in das Textfeld der Software einzugeben. Nun sollten sie die Funktionen der Software ausprobieren, ohne auf die Performance der Software und die Berechnungen zu achten. Am Ende der Studie füllten die Testpersonen den oben genannten Fragebogen aus.

Kapitel 7

Ergebnisse

In diesem Kapitel werden die Ergebnisse der Studie vorgestellt. Dabei werden sowohl die aus den Daten des Experiments und dem Fragebogen abgeleiteten Ergebnisse als auch die Gründe für die während des Experiments aufgetretenen Leistungsprobleme analysiert.

7.1 Probanden

Zu Beginn des Abschlussfragebogens wurden Fragen zur eigenen Person und den eigenen Erfahrungen mit Kommunikations- und Schreibassistenzwerkzeugen gestellt. Wie bereits in Abschnitt 6.5 beschrieben, wurden 6 Probanden für die Studie rekrutiert. Alle Probanden waren männlich und zwischen 24 und 33 Jahre alt. Fünf der sechs Probanden sind in der Softwareentwicklung tätig, ein Proband ist wissenschaftlicher Mitarbeiter in der Informatik. Wie aus der Abbildung 7.1 hervorgeht, nutzen alle Teilnehmer Messenger-Dienste wie Microsoft Teams regelmäßig im beruflichen Kontext. Auch mit Schreibassistenzsystemen hat ein Großteil der Teilnehmer Erfahrung. Fünf Personen gaben zumindest mittlere Erfahrungen an.

Die Programmiererfahrung mit der Programmiersprache Java fiel bei den Teilnehmern unterschiedlich aus. Hier gab es sowohl einen Probanden mit gar keiner Erfahrung als auch einen Probanden mit *eher viel Erfahrung*. Mit drei Teilnehmenden hatte die Mehrheit *mittlere Erfahrung* mit Java als Programmiersprache.

7.2 Nutzungshäufigkeit der Sentimentanalyse

Die Häufigkeit der Nutzung der Sentimentanalysefunktionen wird von den Testpersonen sehr unterschiedlich eingeschätzt. So haben nach eigener Einschätzung drei der Testpersonen die Ergebnisse der Sentimentanalyse nicht beachtet, während zwei der Testpersonen dies bejahten (siehe Abbildung 7.2). Ähnliches gilt für das Anklicken des negativ markierten Satzes

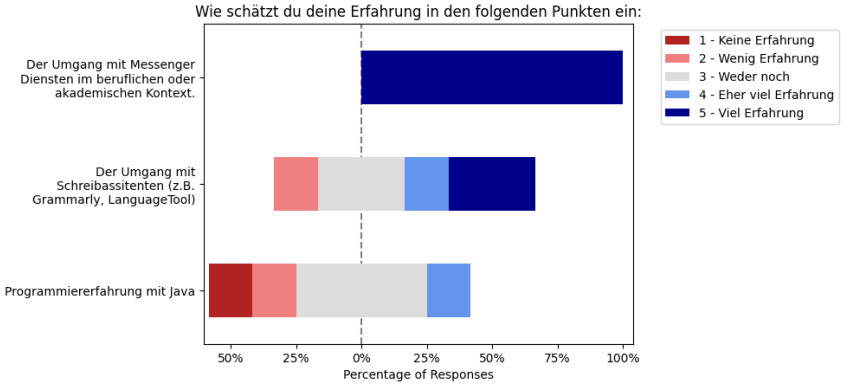


Abbildung 7.1: Selbsteinschätzung der Testpersonen

und die Verwendung des umformulierten positiveren Satzes. Lediglich die Rechtschreibprüfung fällt hier ab. Nur ein Proband stimmt der Aussage zumindest eher zu, die Rechtschreibprüfung häufig genutzt zu haben. Dem stehen vier Teilnehmende gegenüber, die eher nicht oder gar nicht zustimmen.

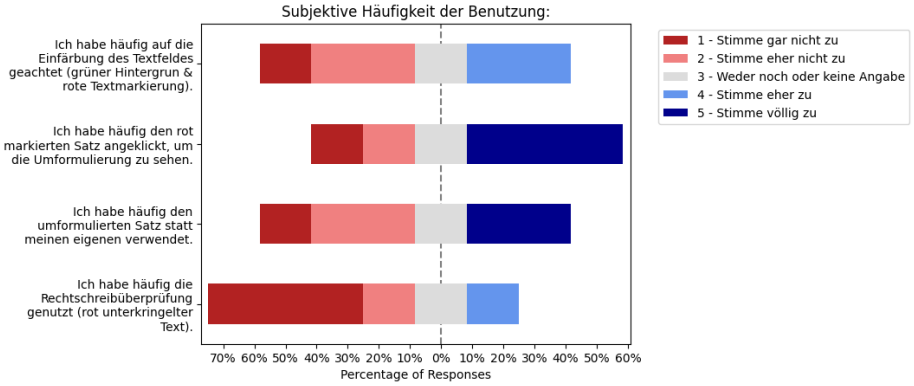


Abbildung 7.2: Einschätzung der Nutzungshäufigkeit der Sentimentanalyse-funktionen

7.3 Bewertung der Sentimentanalyse

Die Studienteilnehmenden wurden nicht nur nach der Häufigkeit der Nutzung gefragt, sondern auch danach, wie sie die Qualität der Sentimentanalyse einschätzen.

Zuerst wurde in Erfahrung gebracht, ob die Anzeige der Stimmung im Textfeld des Messengers als störend empfunden wird. Fünf der Testpersonen gaben hier *trifft überhaupt nicht zu* oder *trifft eher nicht zu* an. Darüber

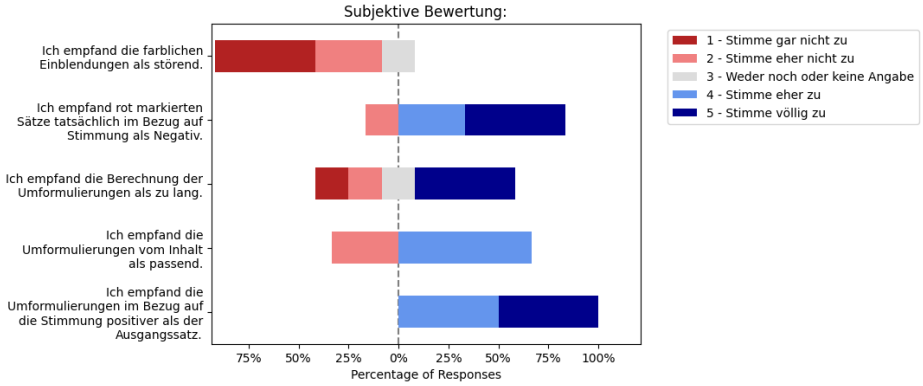


Abbildung 7.3: Einschätzung der Qualität der Sentimentanalysefunktionen

hinaus wurde im Freitextfeld des Fragebogens überwiegend angegeben, dass die Kennzeichnung als nicht störend und angemessen bewertet wurde. Die Testpersonen gewöhnten sich schnell an die Art der Markierung und konnten diese direkt mit der Bedeutung verknüpfen. Als alternative Markierungsmöglichkeit für negative Stimmungen in einem Satz wurden Emojis angegeben. Im Gegensatz zur direkten Markierung des negativen Satzes sollte ein Emoji in der Ecke anzeigen, ob die gesamte Nachricht negative Stimmung enthält oder nicht. Diese Idee wurde damit begründet, dass die Monitore im Versuchsraum die rote Markierung teilweise sehr dunkel darstellten und somit der Text der Nachricht schwer zu lesen war.

Auch die Ergebnisse der Sentimentanalyse (die rote Markierung) wurden von den Studienteilnehmenden überwiegend als zutreffend bewertet. So stimmten fünf der Teilnehmenden der Aussage „Ich empfand die rot markierten Sätze tatsächlich im Bezug auf die Stimmung als Negativ“ eher zu beziehungsweise völlig zu. Während die Ergebnisse für die meisten Testpersonen nachvollziehbar waren, gab ein Proband an, dass nach seinem Empfinden zu wenige Meldungen als negativ identifiziert wurden. So seien z.B. bewusst aggressiv formulierte Nachrichten von der Stimmungsanalyse nicht erkannt worden.

Die Dauer der Berechnung der Umformulierung nach dem Ergebnis der Sentimentanalyse wurde von drei Teilnehmern als zu lang empfunden.

Während vier der sechs Testpersonen die Umformulierung als eher passend und zwei als eher unpassend empfanden, waren sich alle Testpersonen einig, dass die Umformulierung in Bezug auf die hervorgerufene Stimmung positiver ist als der ursprüngliche Satz. Drei Testpersonen gaben hier *Stimme eher zu* und drei *Stimme völlig zu* an. Als Grund dafür, dass die Teilnehmenden bestimmte Umformulierungen im Experiment nicht verwendeten, gaben die Teilnehmenden vor allem an, dass die Ladezeiten zu

lang waren.

Weiter wurde geschildert, dass der negative Ton des Ausgangssatzes dem Autor entweder egal oder sogar gewollt war und deshalb einzelne generierte Umformulierungen nicht übernommen wurden.

7.4 Bewertung der weiteren Funktionen

Da neben der Sentimentanalyse und der Umformulierung von negativen Sätzen weitere Funktionen des Prototyps erarbeitet wurden, beinhaltete der Fragebogen auch Fragen zu diesen. Die subjektive Bewertung der Häufigkeit der Nutzung zeigt Abbildung 7.4.

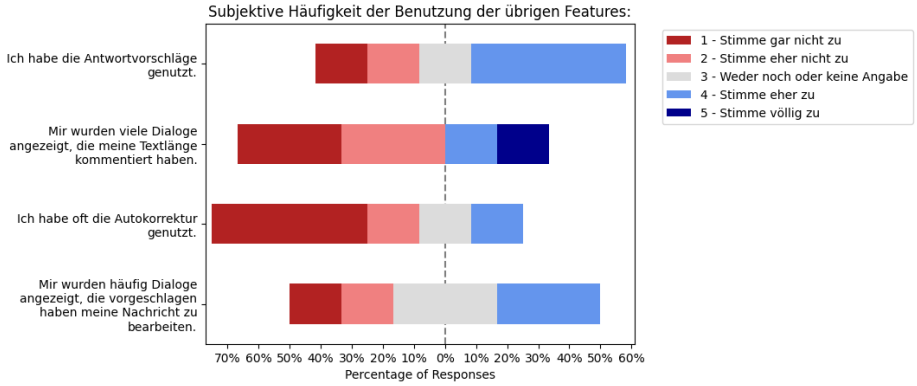


Abbildung 7.4: Einschätzung der Nutzungshäufigkeit der weiteren Funktionen

Hier zeigt sich ähnlich wie bei der Häufigkeit der Nutzung der Sentimentanalysefunktionen ein ausgeglichenes Bild. Die automatischen Antwortvorschläge wurden mit drei Angaben von *Stimme eher zu* auf die Aussage „Ich habe die Antwortvorschläge häufig genutzt.“ am meisten genutzt.

Abbildung 7.5 zeigt die Angemessenheit der weiteren Handlungsempfehlungen. Die generierten Antwortvorschläge wurden von drei Teilnehmenden als *eher nicht passend* und wiederum von drei Teilnehmenden als *eher passend* bewertet. Die Dialoge, die auf einen großen Textlängenunterschied zwischen Nachricht und Antwort aufmerksam machen, wurden von drei Teilnehmenden mit mindestens *Stimme eher nicht zu* als weniger passend empfunden. Während die automatische Bearbeitung von Nachrichten mit sofortiger Verbesserung durch eine zweite Nachricht größtenteils weder als besonders passend noch als besonders unpassend bewertet wurde, wurde die Angemessenheit der Rechtschreibkorrektur nur von einem Teilnehmenden als eher unpassend beurteilt.

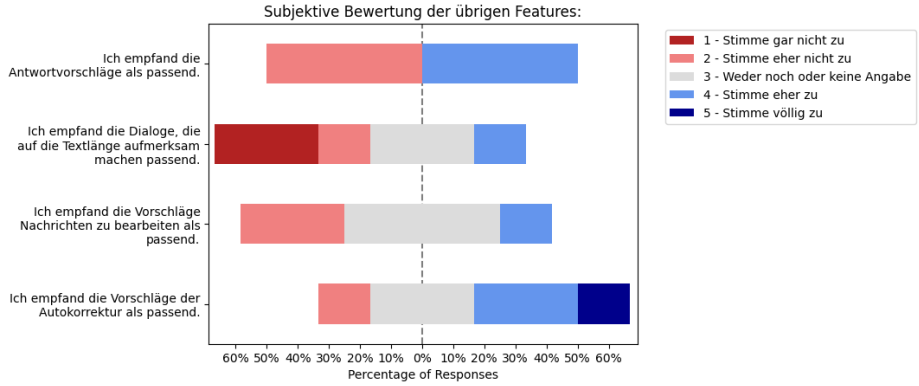


Abbildung 7.5: Einschätzung der Qualität der weiteren Funktionen

7.5 Einsatz des Prototypen in einem Entwicklungsteam

In den Fragen zur Einsetzbarkeit des Prototyps in einem Entwicklungsteam sollten die Testpersonen beantworten, inwieweit sie sich den Nutzen des Prototyps in einem realen Entwicklungsteam vorstellen können. Die jeweiligen Fragen und Antworten sind in Abbildung 7.6 dargestellt.

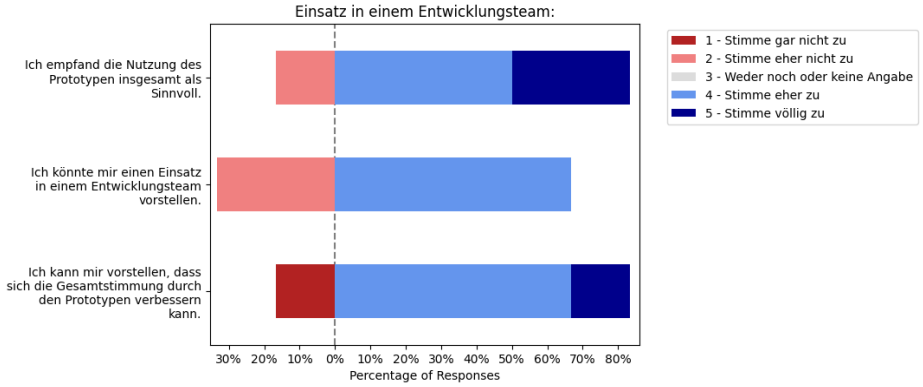


Abbildung 7.6: Bewertung des Einsatzes des Prototypen in einem Entwicklungsteam

Insgesamt wurde der Einsatz des Prototyps als sinnvoll bewertet. Von den sechs Teilnehmenden des Experiments stimmten fünf der Aussage eher oder völlig zu. Auch könnten sich zwei Drittel der Testpersonen sich den Einsatz einer vergleichbaren Software in einem Entwicklungsteam vorstellen. Zuletzt wurde in dieser Fragengruppe gefragt, ob sich die Teilnehmer vorstellen könnten, dass sich durch den Einsatz des Prototyps die Gesamtstimmung in einem Entwicklungsteam verbessern könnte. Vier

der Teilnehmer stimmten dieser Aussage eher zu. Einer stimmte völlig zu und ein Teilnehmer stimmte der Aussage gar nicht zu.

Weiter wurden die Studienteilnehmenden gefragt, was sie an den Prototypen für den Einsatz in einem realen Entwicklungsteam verändern oder verbessern würden. Wie bereits oben erwähnt, wünschen sich die Studienteilnehmenden eine bessere Performance bei der Berechnung der Sentimentanalyse und der Umformulierung. In einem realen Einsatz müsste die Berechnung innerhalb weniger Sekunden abgeschlossen sein, da die Mitarbeiter:innen in ihrer alltäglichen Arbeit keine Zeit durch den Prototyp verlieren sollen.

Neben der Berechnungszeit wurde von den Testpersonen auch der Schreibstil der Umformulierungen thematisiert. Dieser wurde im Kontext des Experiments als zu formal empfunden. So wurde die Idee geäußert, dass das Large Language Model vor dem Einsatz zusätzlich auf die individuellen Stile der Nutzenden trainiert werden könnte. Ein anderer Teilnehmer schlug vor, den Nutzenden in der Benutzeroberfläche die Wahl zu lassen, ob die Umformulierungen eher formal oder informal sein sollen.

7.6 Usability

Neben den Ergebnissen zu der generellen Funktionsweise der Software wurden auch Daten zu der generellen Bedienbarkeit erhoben. Wie in Abschnitt 6.4 beschrieben, wurden den Probanden zusätzlich 10 Aussagen aus dem Katalog des System Usability Scales (SUS) vorgelegt, deren Inhalt sie auf einer Likert-Skala zustimmen oder ablehnen konnten. Im Anhang Abschnitt C.2 sind alle Fragen und die Verteilung der Antworten aufgeführt. Die Fragen sind so gestellt, dass abwechselnd ein positiver und ein negativer Aspekt abgefragt wird, um sicherzustellen, dass die Fragen sorgfältig gelesen und beantwortet werden. Mit dem entsprechenden Faktor -1 für die negativ gestellten Fragen kann ein Score berechnet werden. Dieser geht, wenn alle 10 Fragen berücksichtigt werden, von 0 bis 100. Der Score des vorgestellten Prototyps liegt im Durchschnitt bei 82,5 Punkten.

Kapitel 8

Diskussion

In diesem Kapitel werden zunächst die im Rahmen der Studie erhobenen Kapitel 7 diskutiert und die Leitfragen der Studie (siehe Abbildung 6.1) beantwortet. Dabei werden auch die im Experiment aufgetretenen Performance-Probleme analysiert. Im Schlussteil dieses Kapitels werden die in Kapitel 1 aufgestellten Forschungsfragen beantwortet und Empfehlungen für die weitere Forschung gegeben. Dabei werden auch Lösungsansätze für die Performance-Probleme aufgezeigt und erläutert.

8.1 Nutzen der grafischen Anzeige der Stimmungspolarität

Zunächst soll der generelle Nutzen der grafischen Darstellung von Stimmungen in einem Kommunikationstool analysiert werden. Dabei werden sowohl positive Aspekte als auch mögliche Nachteile diskutiert.

Leitfrage 1. Was hat das grafische Anzeigen der Polarität der Stimmung für Auswirkungen auf den Nutzer?

Bei der subjektiven Bewertung der Sentimentanalyse stimmte kein Proband der Aussage eher oder völlig zu, dass die farbliche Markierung störend war. Dabei wurde die Rotfärbung von negativ erkannten Sätzen und die leichte Grünfärbung des gesamten Textfeldes bei neutraler oder positiver Stimmung berücksichtigt. Demnach lässt sich sagen, dass die Live-Sentimentanalyse im Allgemeinen bei dem Auszug an Testpersonen gut ignoriert werden kann und die Bedienbarkeit der Software nicht negativ beeinflusst. In Bezug auf die in Abschnitt 4.3 aufgestellten Faktoren von Murphy und Murphy-Hill [24] für Empfehlungssysteme deutet dieser Umstand auf eine geringe Ablenkung durch das System hin.

Weiter wurden die Ergebnisse der Sentimentanalyse von fast allen

Probanden als zutreffend bewertet. Das bedeutet, dass die Nutzenden die von der Sentimentanalyse als negativ bewerteten Sätze auch selbst als negativ bewerteten. Dies ist für den Einsatz der Software sehr wichtig, da es das Vertrauen in die Funktionsweise und die Transparenz der Software fördert. Dies erhöht den Nutzen der Software als Empfehlungssystem. In der Regel wurden die von der Software rot markierten Sätze auch angeklickt, um die generierte Umformulierung zu sehen. Die Gründe, warum die Markierung nicht angeklickt wurde, waren in erster Linie performancebedingt. Die Tatsache, dass ein Proband die Umformulierung teilweise nicht sah, weil er seine Nachricht bewusst in einem scharfen Ton schreiben wollte, zeigt, dass die Funktionsweise des Systems schnell verstanden wurde und dass diese auch nicht ablenkt, wenn der/die Nutzende die Empfehlungen und Hinweise nicht benötigt.

8.2 Nutzen der automatischen positiveren Umformulierung

Nun soll die Funktion des automatischen Umformulierungsvorschlags von durch die Sentimentanalyse negativ klassifizierten Sätzen anhand der vorliegenden Ergebnisse bewertet werden.

Leitfrage 2. Wie hoch ist der Nutzen der automatischen Umformulierungen?

Die Umformulierungen wurden von den Probanden mehrheitlich als inhaltlich angemessen bewertet. Insbesondere die Tonalität und Stimmung der umformulierten Sätze wurde von allen Probanden im Vergleich zum Ausgangssatz als positiver bewertet. Das Ziel der Umformulierung, die subjektive Stimmung des Satzes zu verbessern, wurde somit für die Personengruppe, die an der Studie teilgenommen hat, erfüllt. Die Häufigkeit, mit der diese Funktion von den Nutzenden genutzt wird, lässt sich mit den Ergebnissen der Studie leider nicht analysieren. Dies liegt daran, dass die hohe Berechnungszeit der Umformulierung die Nutzung umständlich und langwierig machte. Die geplante realistische Abbildung eines Einsatzes der Software in einem Entwicklungsteam konnte mit den technischen Möglichkeiten leider nicht bewerkstelligt werden.

Leitfrage 3. Wie hoch ist der Nutzen der weiterführenden Funktionen im Hinblick auf die Stimmung?

Die Ergebnisse aus Abschnitt 7.4 zeigen, dass die weiteren Funktionen im Mittel neutral bewertet wurden. Darunter fallen die automatischen Antwortvorschläge, die Hinweise auf Längenunterschiede zwischen Nachricht

und Antwort, das automatische Editieren von Nachrichten und die Rechtschreibkorrektur. Im Gegensatz zu den Funktionen zur Stimmungsanalyse zeigt sich, dass für die Stimmung in der schriftlichen Kommunikation vor allem die Formulierung und die Wahl der Wörter relevant sind. Eine Software, die die Stimmung in eine positive Richtung beeinflussen soll, müsste also direkt am Inhalt ansetzen, anstatt Rahmenbedingungen wie die Länge der Nachricht vorzugeben. Auch die im Vergleich seltenere Nutzung der Rechtschreibkorrektur zeigt, dass es für die Stimmung wichtiger ist, wie eine Nachricht semantisch aufgebaut ist, als dass sie syntaktisch korrekt ist. Die Ergebnisse der Rechtschreibüberprüfung wurden dennoch eher als passend bewertet.

8.3 Beantwortung der Forschungsfragen

Im Folgenden werden die in Kapitel 1 aufgestellten Forschungsfragen dieser Arbeit beantwortet. Dazu werden sowohl die Ergebnisse der Laborstudie als auch die Erkenntnisse aus dem Workshop (vgl. Abschnitt 4.2) und der Implementierung (vgl. Kapitel 5) herangezogen.

Zunächst war es das Ziel, Auslöser für die Verbreitung negativer Stimmung in der textbasierten Kommunikation in Entwicklungsteams zu identifizieren.

FF1

Welche Einflussfaktoren führen zu negativer Stimmung bei der textbasierten Kommunikation in Entwicklungsteams?

In dem durchgeführten Workshop wurden verschiedene Probleme identifiziert, die sich auf die Stimmung der Nutzer auswirken. Dabei wurden bewusst Teilnehmende rekrutiert, die über Erfahrungen im Bereich Software Engineering und der Nutzung von Kommunikationsplattformen im beruflichen Kontext vorweisen können. In dem Workshop wurde allem voran festgestellt, dass schlechte Stimmung über die Sprache zwischen Personen übertragen wird. Anhand von Formulierungen, der An- oder Abwesenheit von Emoticons und der generellen Länge von Nachrichten. Ein weiterer Faktor ist der thematische Bezug der Nachrichten. Wenn Nachrichten nicht zum Inhalt der vorherigen Nachrichten passen oder Sachverhalte diskutiert werden, die zu viele Nachrichten erfordern, fühlen sich die Nutzenden gestört. Als weitere Faktoren wurden Rechtschreibfehler, zu lange Antwortzeiten, die große Menge an Nachrichten und der Zeitaufwand, der teilweise für die Kommunikation aufgewendet werden muss und von der Hauptaufgabe der Entwickelnden ablenkt, identifiziert. Während im

Workshop alle Faktoren mit ähnlicher Priorität behandelt wurden, zeigt die durchgeführte Studie, dass von den umgesetzten Assistenzsystemen diejenigen, die die semantische Bedeutung und die Formulierung betreffen, häufiger genutzt und als sinnvoller bewertet werden als diejenigen, die sich auf die äußere Form der Nachricht beziehen. Dazu zählen zum Beispiel die Länge oder die korrekte Rechtschreibung der Nachrichten.

FF2

Wie lässt sich ein Assistenzsystem für die Stimmung in einem Entwicklungsteam umsetzen und welche aktuellen Technologien sind dafür geeignet?

Nachdem im oben beschriebenen Workshop die Faktoren für negative Stimmungen identifiziert wurden, wurden auch die Möglichkeiten der Einflussnahme auf diese Faktoren von Seiten der Kommunikationsplattform diskutiert. Analog dazu wird in Kapitel 2 auf aktuelle Tools aus den Bereichen *Natural Language Processing* und Sentimentanalyse eingegangen. Um negative Stimmungen auf der inhaltlichen Ebene zu vermeiden und gleichzeitig positive Stimmungen zu fördern, ist ein Ansatz, die geschriebene Nachricht vor dem Versenden entsprechend der Stimmung rot oder grün einzufärben. Um die Polarität der Stimmung zu klassifizieren, eignen sich aktuelle Sentimentanalyse-Tools. In Abschnitt 2.1 wurden verschiedene Werkzeuge vorgestellt. Dazu zählen sowohl lexikonbasierte Tools wie *SentiStrength* als auch weiterführende Tools wie die Sentimentanalyse mit einem Sprachmodell. In dieser Arbeit wurden das Sprachmodell BERT und deren Möglichkeiten zur Sentimentanalyse beschrieben und in der Implementierung des Prototyps genutzt. Neben der farblichen Hervorhebung von Stimmungen wurde auch ein Ansatz zur automatischen Umformulierung negativer Nachrichten analysiert. Dazu kann ein sogenanntes *Large Language Model*, wie in Kapitel 2 beschrieben, verwendet werden. Die Möglichkeiten und die Nutzbarkeit für Projekte mit begrenzten Ressourcen haben sich in den letzten Jahren rasant weiterentwickelt, so dass auch komplexere Aufgaben des *Natural Language Processing*, wie die Umformulierung von Texten, gelöst werden können. In dieser Arbeit wird dazu das Open-Source-Modell LLaMA verwendet. Zusätzlich zu den beiden Funktionen zur Stimmungsanalyse wurden in der Kapitel 5 auch Methoden des maschinellen Lernens zur Generierung von Antwortvorschlägen eingesetzt. Ziel dieses Assistenten ist es, die durchschnittliche Antwortzeit zu verkürzen und den Kommunikationsaufwand zu verringern.

Die Beachtung von Rahmenbedingungen wie Textlänge und Rechtschreibüberprüfung sind Aufgaben, die im *Natural Language Processing* bereits gelöst sind. Hierbei bedarf es keiner neuen technischen Ansätze.

Für die Rechtschreibanalyse wurde das Werkzeug Hunspell verwendet, das in weit verbreiteten Produkten wie Mozilla Firefox oder den OpenOffice-Programmen eingesetzt wird.

FF3

Wie lässt sich das Konzept in der Praxis in Entwicklungsteams einsetzen?

Die Nutzerstudie hat gezeigt, dass die Auswahl der Nutzenden, die bereits Erfahrungen mit der Arbeit in Entwicklungsteams haben, den Einsatz des Prototyps in diesem Kontext im Mittel als sinnvoll bewerten. Sie konnten sich mehrheitlich einen Einsatz des Prototypen in einen Entwicklungsteam vorstellen. Der Prototyp hat nach Ansicht der Befragten auch das Potenzial, die Gesamtstimmung in einem Entwicklungsteam zu verbessern. Anhand dieser Ergebnisse kann die Frage dahingehend beantwortet werden, dass sich das Konzept gut in einem Entwicklungsteam einsetzen lässt und das Ziel, eine positive Stimmung zu fördern, erfüllen kann. Andererseits hat die Auswertung der Studie zusätzliche Aspekte und Faktoren aufgezeigt, die für den Einsatz des Prototyps wichtig sind.

Dies sind etwa Fragen zu der Performance des Assistenzsystems. Da die Berechnungsgeschwindigkeit in der Studie nicht im erwarteten Rahmen lag, wurden die Funktionen der Software häufig ignoriert. Für den Einsatz der Software bedeutet dies, dass eine entsprechend leistungsstarke Infrastruktur vorhanden sein muss, um das volle Potenzial des Prototyps auszureizen. Aus der Nutzersicht ließe sich der Prototyp hingegen leichter in den Arbeitsalltag integrieren.

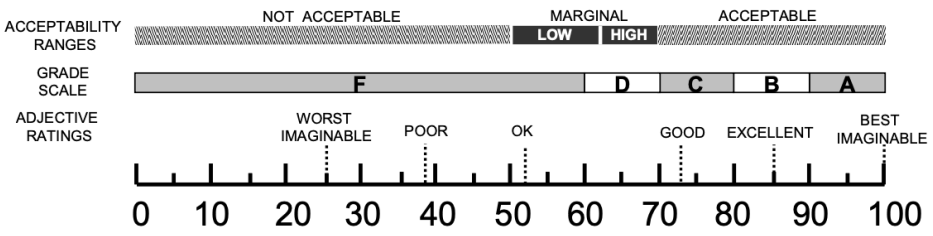


Abbildung 8.1: Skalen zu der Bewertung von System Usability Scale Punktzahlen (aus Bangor et al. [6])

Bei dem Prototyp dieser Arbeit wurde an der generellen Funktionsweise einer branchentypischen Kommunikationsplattform nichts verändert und das Assistenzsystem lediglich zusätzlich zum gewohnten Funktionsumfang

eingebettet. Ein grundsätzlicher Wechsel der Plattform oder eine größere Umgewöhnung auf Seiten der Nutzer ist daher nicht notwendig. Die zusätzlichen Sentimentanylse-Funktionen wirkten zudem nicht störend oder ablenkend. Zudem weist das System nach der *System Usability Scale* mit einem Wert von 82,5 Punkten einen überdurchschnittlichen Wert auf. Ein Wert von 68 bildet hier die Mitte als Referenz. Mit der erreichten Punktzahl des Prototyps liegt dieser nach Abbildung 8.1 zwischen den subjektiven Bewertungen *gut* und *exzellent*. Besonders die Aspekte der einfachen Bedienung und der schnellen Beherrschbarkeit sind für den Einsatz in der Praxis entscheidend.

8.4 Performance

Nun sollen die Probleme, die bei der Durchführung der Studie mit dem Prototyp aufgetreten sind, analysiert und Lösungsansätze vorgeschlagen werden.

Im Rahmen der Implementierung und Vorbereitung der Studie wurde im Selbsttest eine Verarbeitungszeit der Anfragen an das BERT-Modell zur Stimmungsklassifikation von unter einer Sekunde festgestellt. Die Umformulierung mit dem LLaMA-Modell dauerte je nach Satzlänge auf der verfügbaren Hardware etwa 5 bis 12 Sekunden. Diese Berechnungszeiten wurden im Hinblick auf den Prototypenstatus vom Verfasser dieser Arbeit als annehmbar bewertet. So hat auch Nielsen [27] 10 Sekunden als maximal zumutbare Wartezeit in einem Softwaresystem aufgestellt. Die Funktionen der Software hätten also im Rahmen der Studie sinnvoll evaluiert werden können. Ob in dem konkreten Fall die Wartezeit von 5 bis 12 Sekunden zu lange gewesen wäre, sollte in der Studie ebenfalls evaluiert werden (siehe Fragebogen im Anhang C). Für den Einsatz in einem realen Entwicklungsteam müsste überprüft werden, inwieweit diese Berechnungsdauer durch eventuell stärkere Hardware reduziert werden könnte.

In der Studie haben sich diese Berechnungszeiten (5 bis 12 Sekunden) jedoch als zu wenig skalierbar herausgestellt. Grund dafür ist, dass die meisten Bibliotheken aus dem Bereich des maschinellen Lernens, so auch die verwendete *Transformer*-Bibliothek von Huggingface, darauf ausgelegt sind, alle zur Verfügung stehenden Ressourcen der Hardware optimal für die Berechnung zu nutzen [35]. So erhöht sich die Latenz drastisch, wenn die Anzahl der Clients und damit die Anzahl der Anfragen steigt. Die Anfragen werden stets nacheinander bearbeitet. Dies wäre prinzipiell kein Problem, wenn die Zeit, die die Modelle zur Vorhersage benötigen, vorhersagbar und konstant wäre. Dies ist jedoch aufgrund der Natur von KI-Modellen nicht der Fall. Zudem gibt es im Bereich des Deep Learnings das Phänomen des *Überdenkens*. [19]. Dieses besagt, dass häufig weiter an einer Vorhersage

gerechnet wird, obwohl sich das Ergebnis nicht mehr verändert. Das führt zu einem nicht optimalen Einsatz der Ressourcen.

Wenn nun die Berechnung einzelner Anfragen mehr Zeit als gewöhnlich in Anspruch nimmt, verzögert sich die Bearbeitung nachfolgender Anfragen anderer Clients. Dies führt zu einer Art Kettenreaktion, so dass die Python-Anwendung mit den Antworten nicht mehr hinterherkommt. Diese Verarbeitung zeigt Abbildung 8.2.

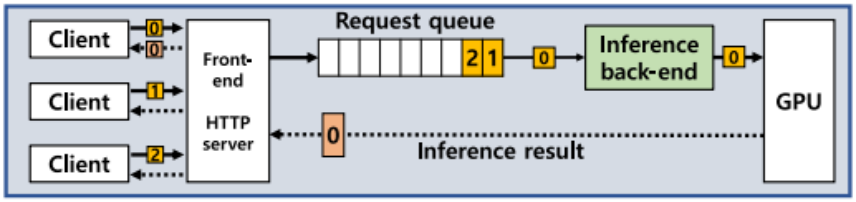


Abbildung 8.2: Ablauf der Anfragenverarbeitung für ein einfaches Model Backend (aus Ahn et al. [1])

Um diesen Problemen entgegenzuwirken, gibt es verschiedene Ansätze. Zum einen das sogenannte *Batching*. Hierbei werden mehrere Eingaben zusammengefasst und gleichzeitig vom Modell verarbeitet. Dies ist in der Regel effizienter, als die Operationen einzeln auszuführen. Eine weitere Möglichkeit ist das *Multiprocessing*. Dabei werden die Ressourcen nicht auf eine Verarbeitung konzentriert, sondern auf mehrere parallele Berechnungen verteilt. Es werden separate Instanzen des Modells erzeugt, die auf den einzelnen Kernen der CPU laufen und über die API angesprochen werden können.

Neben der Optimierung der Berechnungen auf der Backend-Seite könnte andererseits auch die in Abschnitt 5.2 vorgestellte Architektur angepasst werden. Aufgrund der vergleichsweise geringen Größe des BERT-Modells und der einfachen Berechnungen für die weiteren Funktionen der Software bietet es sich an, diese auf die lokalen Maschinen der Clients zu verlagern. Diese benötigen nicht die Rechenleistung eines Servers und können daher auch auf lokalen Rechnern betrieben werden. Dadurch erhöht sich zwar die Rechenzeit geringfügig, aber insgesamt wäre das System mit dieser Lösung für viele Clients besser skalierbar. Diese Lösung ist jedoch nicht für die Umformulierung geeignet, da das für die Berechnungen verantwortliche Modell nur auf leistungsstarken GPUs effizient läuft.

8.5 Limitierungen

In diesem Abschnitt wird die Gültigkeit der Ergebnisse eingeordnet. Dazu werden die Ergebnisse anhand der von Wohlin et al. [41] aufgestellten

Aspekte der Validität überprüft. Dies dient dazu, den Rahmen der Gültigkeit der Ergebnisse einzuschränken. Weiter wird aufgezeigt, welche Fragen das Experiment nicht beantwortet.

8.5.1 Bedrohungen der Validität

Bedrohungen der ***Conclusion Validity*** beschreiben Einflussfaktoren, die das Ziehen von korrekten Schlussfolgerungen einschränken. Im Falle dieser Arbeit könnte sie durch die geringe Teilnehmerzahl der Studie gefährdet sein. So könnten die Ergebnisse in einer größeren Versuchsgruppe von den Ergebnissen dieser Arbeit abweichen. Der experimentelle Charakter der Studie gefährdet auch die *Reliability of measures*. Dieses Prinzip besagt, dass die Messung von Phänomenen wiederholbar sein muss. Da viele verschiedene soziale und zwischenmenschliche Faktoren in das Experiment involviert waren, kann es sein, dass sich die Teilnehmenden bei einer Wiederholung anders verhalten und zu anderen Ergebnissen kommen. Hierbei ist allerdings festzustellen, dass eine *Reliability of measures* aufgrund des Forschungsgegenstands, der Stimmung, ohnehin nur sehr schwer möglich ist.

Bedrohungen der ***Internal Validity*** sind Einflüsse, die die Rückschlüsse auf einen kausalen Zusammenhang betreffen können. Durch das auf die Anwendbarkeit des Prototyps ausgerichtete Studiendesign gab es keine Kontrollgruppe. Vielmehr wurde in der präsentierten Studie ein Machbarkeitsnachweis durchgeführt. Dieser gibt Hinweise auf mögliche Zusammenhänge. Um diesen weiter nachzugehen und die Zusammenhänge zu bestätigen müssen weitere Studien durchgeführt werden.

Construct Validity beschreibt die Möglichkeit, von den Ergebnissen des Experiments auf das Konzept oder die Theorie hinter dem Experiment zu schließen. Hier kommen vor allem soziale Bedrohungen ins Spiel. Den Versuchspersonen war bewusst, dass sie an einem Experiment teilnehmen. Daher kann die Echtheit von Stimmungen und Emotionen im Kontext des Experiments in Frage gestellt werden. Auch ist davon auszugehen, dass die Teilnehmer im Kontext eines Experiments grundsätzlich freundlicher zueinander sind, da ihnen der emotionale Bezug zur Aufgabe fehlt. Dieses Phänomen war dem Autor in der Konzeption des Experiments bewusst. Aus diesem Grund wurden mit den in Kapitel 6 beschriebenen Faktoren Stressoren implementiert, auf welche die Testpersonen sich nicht im Vorfeld vorbereiten konnten.

Als weitere Bedrohung der Validität kommt hinzu, dass alle Probanden aus dem sozialen Umfeld des Autors dieser Arbeit stammen. Dieser

Umstand kann dazu führen, dass die Probanden bewusst oder unbewusst *Experimenter expectancies* erfüllen. Dies bedeutet, dass die Probanden das Ziel der Studie durchschauen und zum Beispiel Fragen nach den eigenen Erwartungen oder den Erwartungen des Experimentators beantworten. Aus diesem Grund wurde ein Experimentdesign gewählt, bei dem die Versuchspersonen eine primäre Aufgabe erhalten und der zu untersuchende Prototyp nur sekundär genutzt wird.

Die ***External Validity*** wird durch Faktoren beeinflusst, die die Generalisierbarkeit der Ergebnisse einschränken. Wie bereits erwähnt, nahmen an dem Experiment nur Personen aus dem Umfeld des Autors teil. Auch dies kann die ***External Validity*** bedrohen. Um eine generalisierbare Aussage treffen zu können, müssten verschiedene Personengruppen und Stakeholder der Software in die Evaluation einbezogen werden. Auch können die Ergebnisse nicht auf andere Personengruppen mit anderen Aufgaben als im Experiment übertragen werden. Trotzdem konnten mit den Erfahrungen, die die Testpersonen aufweisen konnten, eine Sinnvolle Stichprobe erstellt werden. Auch haben alle Teilnehmer einen Bezug zu dem Thema der Arbeit und konnten mit ihrer Erfahrung und Expertise in den Bereichen Software Engineering und Kommunikationstools sinnvoll den Prototypen evaluieren.

8.5.2 Weitere Limitierungen

Neben den Bedrohungen der Validität soll in diesem Abschnitt auch geklärt werden, über welche Aspekte keine Aussage getroffen werden kann. Im Experiment wurde die Einsetzbarkeit des Prototyps analysiert. Die Ergebnisse sind daher als Hinweis zu verstehen, dass das Konzept dieser Arbeit umsetzbar ist und der erste Test mit Probanden darauf hindeutet, dass eine Stimmungsverbesserung durch den Assistenten möglich ist. Diese Aussage beruht allerdings auf den Erfahrungen und Einschätzungen der Probanden. Ob die Stimmung in einem Entwicklungsteam tatsächlich über einen längeren Zeitraum verbessert werden kann, konnte mit der vorgestellten Studie nicht belegt werden (zu weiteren Forschungsmöglichkeiten siehe Abschnitt 9.2). Weiter wurde die Qualität der Umformulierungen anhand des Empfindens der Teilnehmenden evaluiert. Mögliche inhaltliche Verschiebungen, die zum Beispiel Diskriminierungen beinhalten könnten, können durch die vorgestellte Evaluierung nicht ausgeschlossen werden.

Neben Detailfragen zum Prototyp und der Umsetzung wurde von einer Person die Frage aufgeworfen, inwieweit Menschen bereit sind, sich von einem KI-System beeinflussen zu lassen. Die Person war selbst nicht von der Anwendung von KI bei Alltagsproblemen überzeugt. Wie weit dieses Problem reicht und ob der Einsatz des vorgestellten Prototyps gesellschaftlich akzeptiert wäre, kann ebenfalls nicht beantwortet werden, da der Fokus dieser Arbeit auf den technischen Aspekten liegt. Dennoch ist

die Kritik an der Thematik berechtigt, sodass für einen Einsatz zusätzlich untersucht werden müsste, mit welchen Mitteln das Vertrauen in die Technologie erhöht werden kann und welche Sicherheitsaspekte für die Nutzenden von hoher Relevanz sind und somit Vertrauen schaffen.

Kapitel 9

Zusammenfassung und Ausblick

Im letzten Kapitel dieser Arbeit werden zunächst alle wichtigen Ergebnisse dieser Arbeit zusammengefasst. Abschließend wird ein Ausblick auf mögliche weiterführende Forschung gegeben.

9.1 Zusammenfassung

Die Stimmung ist ein essentieller Faktor für den Erfolg von Softwareprojekten. Eine Vielzahl von Forschungsarbeiten beschäftigt sich bereits mit der Analyse der Stimmung in Entwicklungsteams und der visuellen Aufbereitung dieser Ergebnisse. Der praktische Nutzen dieser Ergebnisse bleibt jedoch abstrakt und es obliegt der eigenen Initiative, ob Maßnahmen zur Verbesserung der Stimmung ergriffen werden oder nicht. Die vorliegende Arbeit geht einen Schritt weiter und analysiert, inwieweit das Kommunikationswerkzeug direkt eine positive Stimmung bei den Nutzern fördern kann. Um **Faktoren** für negative Stimmung in Entwicklungsteams zu **identifizieren**, wurde zunächst ein Workshop mit erfahrenen Softwareentwicklern konzipiert und durchgeführt. Dabei wurden verschiedene Faktoren und Lösungsansätze auf Seiten der Kommunikationsplattform erarbeitet und diskutiert. Nachdem mit Hilfe der Literatur weitere Anforderungen an ein Empfehlungssystem formuliert wurden, wurde ein Konzept für einen Software-Assistenten entwickelt, der direkt im Textfeld eines Messengers die Stimmung live analysiert und bei negativem Ergebnis Verbesserungsvorschläge macht. Zusätzlich sollte die Software eine Rechtschreibüberprüfung bieten, übermäßige Großschreibung und Interpunktion bei der Stimmungsanalyse berücksichtigen, automatische Antwortvorschläge generieren und die Textlänge zwischen Nachricht und Antwort analysieren.

Weiter wurden verschiedene **Ansätze** und **Technologien** vorgestellt, mit denen dieses Konzept implementiert werden kann. Auf Basis einer Open Source Messenger Software wurde mit Hilfe von KI-Sprachmodellen ein Assistenzsystem implementiert, das zunächst die Stimmung in einem Satz analysiert und automatisch eine Umformulierung generiert. Auch die weiteren Funktionen wurden umgesetzt, um auf die in den Workshop herausgearbeiteten Faktoren für negative Stimmung Einfluss zu nehmen.

Um die **praktische Einsetzbarkeit** des Prototypen zu überprüfen, wurde zum Abschluss der Arbeit eine Laborstudie entworfen und durchgeführt, in der ein Szenario eines Softwareprojektes kreiert wird und die Probanden mithilfe der Kommunikation über den Prototypen Programmieraufgaben lösen müssen. Dabei wurden verschiedene Einflüsse implementiert, die für Stress und Spannung bei den Probanden sorgen. Die Ergebnisse weisen darauf hin, dass der gewählte Ansatz zur Implementierung der Funktionen geeignet ist und ein Assistent auf Basis des Prototyps in der Praxis in einem Entwicklungsteam einsetzbar wäre. Dafür sprechen die vergleichsweise gute und einfache Usability der Software und die Tatsache, dass die Probanden insbesondere die Empfehlungen der Stimmungsanalysesysteme als angemessen bewerten. Auch hat die Studie gezeigt, dass ein Einsatz in einem Entwicklungsteam vorstellbar ist und das Potenzial besteht, die Stimmung im Team nachhaltig zu verbessern.

9.2 Ausblick

Neben den Ergebnissen zur Einsetzbarkeit hat die Arbeit zusätzlich weitere Fragen, die untersucht werden müssten, aufgeworfen.

Zunächst müsste überprüft werden, inwieweit die Ergebnisse der Studie auf die Realität übertragbar sind. Dazu wäre eine längerfristige Studie und der Einsatz in einem realen Softwareprojekt notwendig. So könnte auch über einen längeren Zeitraum die tatsächliche Auswirkung auf die Teamstimmung betrachtet werden. Hierbei müssten allerdings zusätzliche Aspekte wie der Schutz von Unternehmensdaten und der Privatsphäre der Nutzenden berücksichtigt werden. Weiter müsste eine sichere Infrastruktur für den Betrieb der Sprachmodelle bereitgestellt werden oder die Architektur so überarbeitet werden, dass die Sprachmodelle lokal auf dem Rechner des/der Anwendenden laufen.

Die Studie hat auch gezeigt, dass es nötig ist, den Stil der Umformulierungen an die individuellen Nutzer anzupassen, um die Nutzungshäufigkeit der Funktion zu erhöhen. Es sollte daher analysiert werden, inwieweit ein Sprachmodell, mit dem positivere Umformulierungen generiert werden können, den Schreibstil des Benutzers berücksichtigen kann. Denkbar wäre

auch eine Funktion, die es dem/der Nutzenden erlaubt, je nach Adressat zwischen einem formellen und einem informellen Schreibstil zu wechseln.

Ein weiterer Aspekt, der im Kontext dieser Arbeit weiter untersucht werden sollte, ist die Auswirkung der Bereitschaft und Motivation der Nutzer, die Stimmung tatsächlich verbessern zu wollen bzw. sich dabei von der Software unterstützen zu lassen. So hat die Studie gezeigt, dass Personen zum Teil bewusst Nachrichten mit negativem Tonalität verfassen. Hier kann untersucht werden, wie das System zusätzlich ein Bewusstsein für die Bedeutung und die daraus resultierenden Gefahren für das Softwareprojekt schaffen kann. Ein anderer Ansatz wäre, negative Nachrichten subtiler umzuformulieren. Beispielsweise könnte ein System entwickelt werden, bei dem die Umformulierungen auf Wunsch des Empfängers vorgenommen werden, so dass sich die bewusste negative Stimmung nicht im Team ausbreiten kann. Bei diesem Ansatz müssten allerdings höhere Anforderungen an die inhaltliche Korrektheit des neuen Satzes gestellt werden, da die menschliche Überprüfung der Qualität der Umformulierung im Vergleich zur Software in dieser Arbeit entfällt.

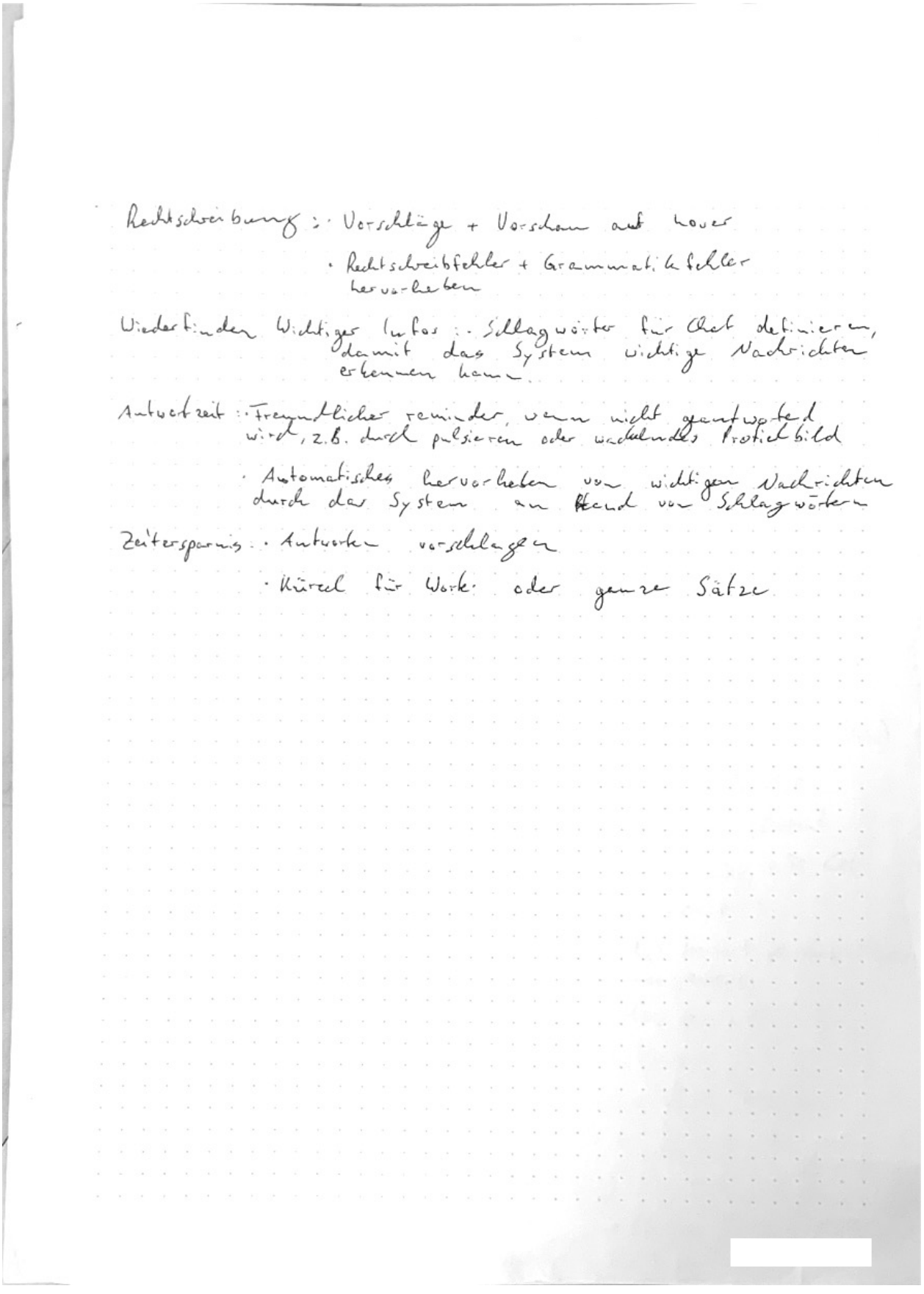
Auch hat das Experiment gezeigt, dass Vertrauen ein kritischer Faktor für die Nutzung von KI-Systemen ist. Hier könnte untersucht werden, welche Anforderungen an die Erklärbarkeit in diesem konkreten Fall gestellt werden müssten, damit das Vertrauen möglichst hoch ist. Außerdem müsste die gesellschaftliche Akzeptanz eines Systems untersucht werden, das indirekt Einfluss auf die Stimmung der Nutzenden nimmt.

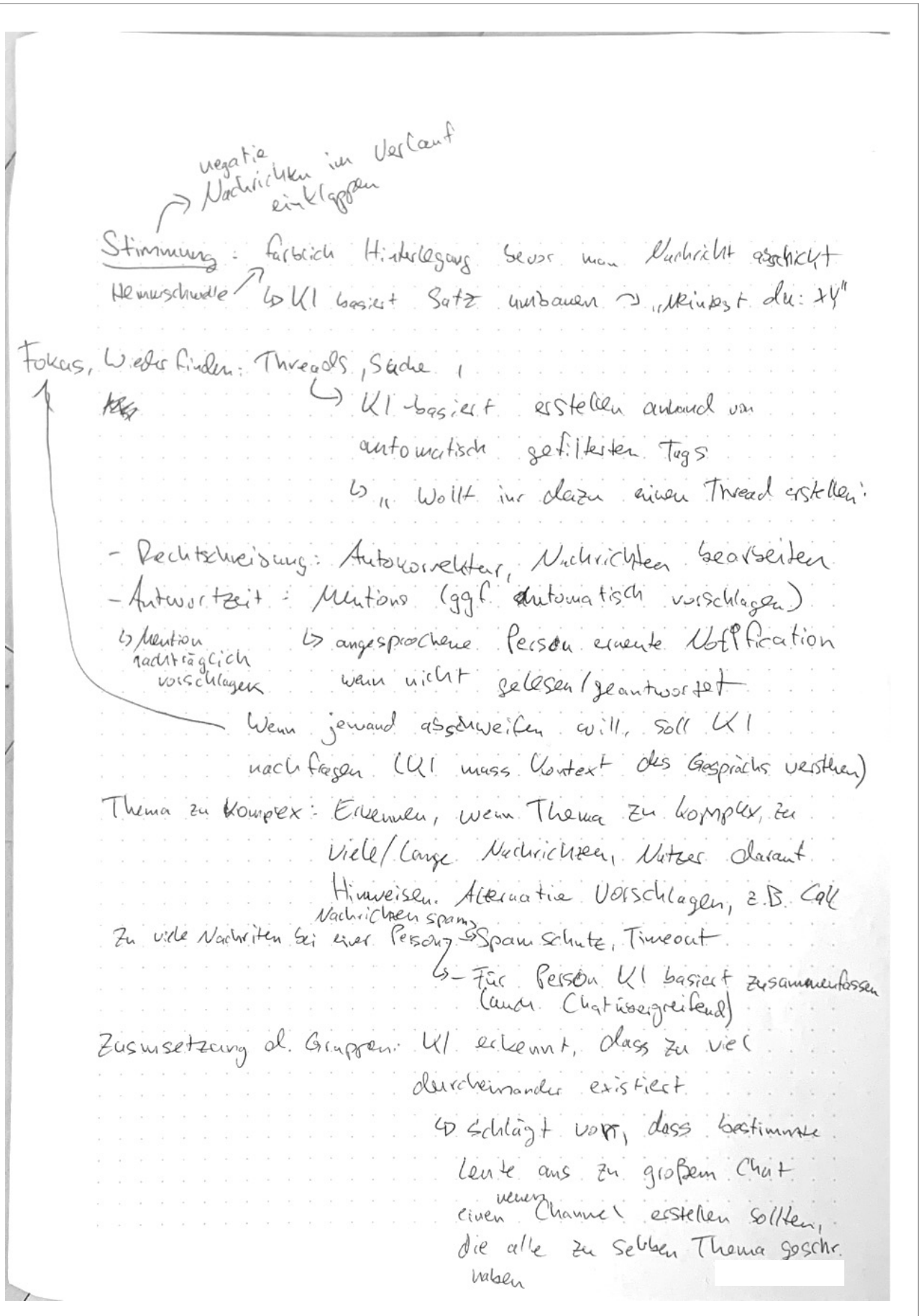
Anhang A

Unterlagen aus dem Workshop

A.1 Schriftliche Ergebnisse zu den Umsetzungen aus dem Workshop

Im Nachfolgenden sind die schriftlichen Ausarbeitungen zu der 3. Leitfrage des Workshops zu sehen.


Rechtschreibung: · Vorschläge + Vorschau auf Lösung
· Rechtschreibfehler + Grammatikfehler hervorheben
Wiederfinden wichtiger Infos: · Schlagwörter für Chat definieren, damit das System wichtige Nachrichten erkennen kann.
Antwortzeit: · freundlicher reminder, wenn nicht geantwortet wird, z.B. durch pulsieren oder wechselndes Profilbild
· Automatisches hervorheben von wichtigen Nachrichten durch das System an Hand von Schlagwörtern
Zeitersparnis: · Antworten vorschlagen
· Kürzel für Worte oder ganze Sätze


negative Nachrichten im Verlauf einklappen
Stimmung: farblich Hinterlegung bevor man Nachricht abschickt
Hemmschwelle
↳ KI basiert Satz umbauen → „Meinst du: xy"
Fokus, Wiederfinden: Threads, Suche
↳ KI-basiert erstellen anhand von automatisch gefilterten Tags
↳ „ Wollt ihr dazu einen Thread erstellen:
- Rechtschreibung: Autokorrektur, Nachrichten bearbeiten
- Antwortzeit: Mentions (ggf. automatisch vorschlagen)
↳ Mention nachträglich vorschlagen
↳ angesprochene Person erneute Notification wenn nicht gelesen/geantwortet
Wenn jemand abschweifen will, soll KI nachfragen (KI muss Kontext des Gesprächs verstehen)
Thema zu komplex: Erkennen, wenn Thema zu komplex, zu viele/lange Nachrichten, Nutzer darauf Hinweisen. Alternative Vorschlagen, z.B. Call
Zu viele Nachrichten bei einer Person → Nachrichten spam
↳ Spam schutz, Timeout
↳ Für Person KI basiert zusammenfassen (auch Chatübergreifend)
Zusmsetzung d. Gruppen: KI erkennt, dass zu viel durcheinander existiert
↳ schlägt vor, dass bestimmte Leute aus zu großem Chat einen neuen Channel erstellen sollten, die alle zu selben Thema geschr. haben

Anhang B

Unterlagen zur Implementierung

B.1 Muster der Antwortmöglichkeiten

Folgende Muster wurden bei der Auswahl der zur Verfügung stehenden Antwortmöglichkeiten verwendet.

```
classes_dict["greeting"] = {}
classes_dict["greeting"]["pattern"] = ["hi", "hallo", "guten morgen", "hey", „guten
nachmittag", "guten abend", "guten tag"]
classes_dict["greeting"]["response"] = ["Hallo 😊", "Wie geht es dir?", "Guten Tag"]

classes_dict["goodbye"] = {}
classes_dict["goodbye"]["pattern"] = ["tschüss", "auf wiedersehen", "bis später“,
"schönen feierabend", "ich muss los", "bis bald", "bis dann"]
classes_dict["goodbye"]["response"] = ["Tschüss", "Auf wiedersehen!"]

classes_dict["thanks"] = {}
classes_dict["thanks"]["pattern"] = ["danke", "dankeschön", "danke dir"]
classes_dict["thanks"]["response"] = ["Gern geschehen!", "Keine Ursache!"]

classes_dict["praise"] = {}
classes_dict["praise"]["pattern"] = ["gut gemacht", "sehr schön", "guter job", „gute
Arbeit geleistet"]
classes_dict["praise"]["response"] = ["Dankeschön!", "Es freut mich, dass ich helfen
konnte"]

classes_dict["how are you"] = {}
classes_dict["how are you"]["pattern"] = ["wie gehts", "wie geht es dir", "was geht"]
classes_dict["how are you"]["response"] = ["Mir geht es gut. Danke!", "War schon mal
besser."]

classes_dict["future"] = {}
classes_dict["future"]["pattern"] = ["wirst du", "kannst du", "würdest du", "machst du"]
classes_dict["future"]["response"] = ["Ja, das kann ich übernehmen", "Nein, ...",
"Vielleicht"]

classes_dict["are you"] = {}
classes_dict["are you"]["pattern"] = ["bist du", "hast du", "warst du"]
classes_dict["are you"]["response"] = ["Nein", "Ja", "Vielleicht"]

classes_dict["you are"] = {}
classes_dict["you are"]["pattern"] = ["du bist", "du hast"]
classes_dict["you are"]["response"] = ["Dankeschön!", "Das Tur mir Leid :("]

classes_dict["when"] = {}
classes_dict["when"]["pattern"] = ["wann wird", "wann kannst", "wann soll", "wann
sollte", "bis wann", "wann ist fertig"]
classes_dict["when"]["response"] = ["Das dauert noch ...", "Kann ich nicht sagen."]

classes_dict["finished"] = {}
classes_dict["finished"]["pattern"] = ["ist fertig", "hier ist", "bin fertig"]
classes_dict["finished"]["response"] = ["Super!", "Danke dir!"]

classes_dict["confirmation"] = {}
classes_dict["confirmation"]["pattern"] = ["ich denke", "ich finde", "ich glaube", "es
ist sinvoll", "es ist wichtig"]
classes_dict["confirmation"]["response"] = ["Ja, genau!", "Richtig!", "Verstanden", „Ich
sehe das anders, ...", "Warum meinst du das?"]
```

```
classes_dict["information"] = {}
classes_dict["information"]["pattern"] = ["kannst du mir sagen", "weißt du"]
classes_dict["information"]["response"] = ["Ja. ...", "Nein das weiß ich nicht."]

classes_dict["who"] = {}
classes_dict["who"]["pattern"] = ["Wer", "wer möchte", "wer kann", "wer hat"]
classes_dict["who"]["response"] = ["Ich", "Ich nicht", "Ich kann das machen"]

classes_dict["invitation"] = {}
classes_dict["invitation"]["pattern"] = ["Hast du Lust", "wie wäre es", "möchtest du"]
classes_dict["invitation"]["response"] = ["Das klingt gut!", "Nein leider nicht"]

classes_dict["sorry"] = {}
classes_dict["sorry"]["pattern"] = ["tut mir leid", "entschuldigung", "ich bedauere“,
"sorry"]
classes_dict["sorry"]["response"] = ["Ist nicht so schlimm. :)", "Das kann passieren“,
"Kein Problem, aber versuche das in Zukunft zu vermeiden"]

classes_dict["request"] = {}
classes_dict["request"]["pattern"] = ["bitte mach", "bitte mache", "mach", "mache", „du
bitte"]
classes_dict["request"]["response"] = ["Wird erledigt", "Alles klar", "Kein Problem"]

classes_dict["criticism"] = {}
classes_dict["criticism"]["pattern"] = ["war schlecht", "ist schlecht", "war nicht gut“,
"ist nicht gut"]
classes_dict["criticism"]["response"] = ["Das tut mir leid.", "Was kann ich besser
machen?"]

classes_dict["welcome"] = {}
classes_dict["welcome"]["pattern"] = ["willkommen", "mein name ist"]
classes_dict["welcome"]["response"] = ["Hallo", "Herzlich Willkommen!"]

classes_dict["idea"] = {}
classes_dict["idea"]["pattern"] = ["was haltet", "was hälst", "Vielleicht sollte",
"Vielleicht sollten"]
classes_dict["idea"]["response"] = ["Das klingt gut", "Ich denke eher, dass ..."]

classes_dict["why"] = {}
classes_dict["why"]["pattern"] = ["warum ist", "warum machst", "wieso machst", "wieso
ist", "weshalb ist", "weshalb machst", "welchem grund", "warum hast du"]
classes_dict["why"]["response"] = ["Das kann ich dir auch nicht sagen.", "Weil …",
"Aufgrund ..."]

classes_dict["how"] = {}
classes_dict["how"]["pattern"] = ["warum ist", "warum machst", "wieso machst", "wieso
ist", "weshalb ist", "weshalb machst", "welchem grund"]
classes_dict["how"]["response"] = ["Das kann ich dir auch nicht sagen.", "Weil ...",
"Aufgrund ..."]
```

Anhang C

Unterlagen der Studie

C.1 Fragebogen

Im Rahmen der Studie wurde zum Ende hin der folgende Fragebogen den Probanden vorgelegt. Dieser erklärt zunächst das Ziel der Studie und die Regeln beziehungsweise die Aufgabenstellung des Experiments.

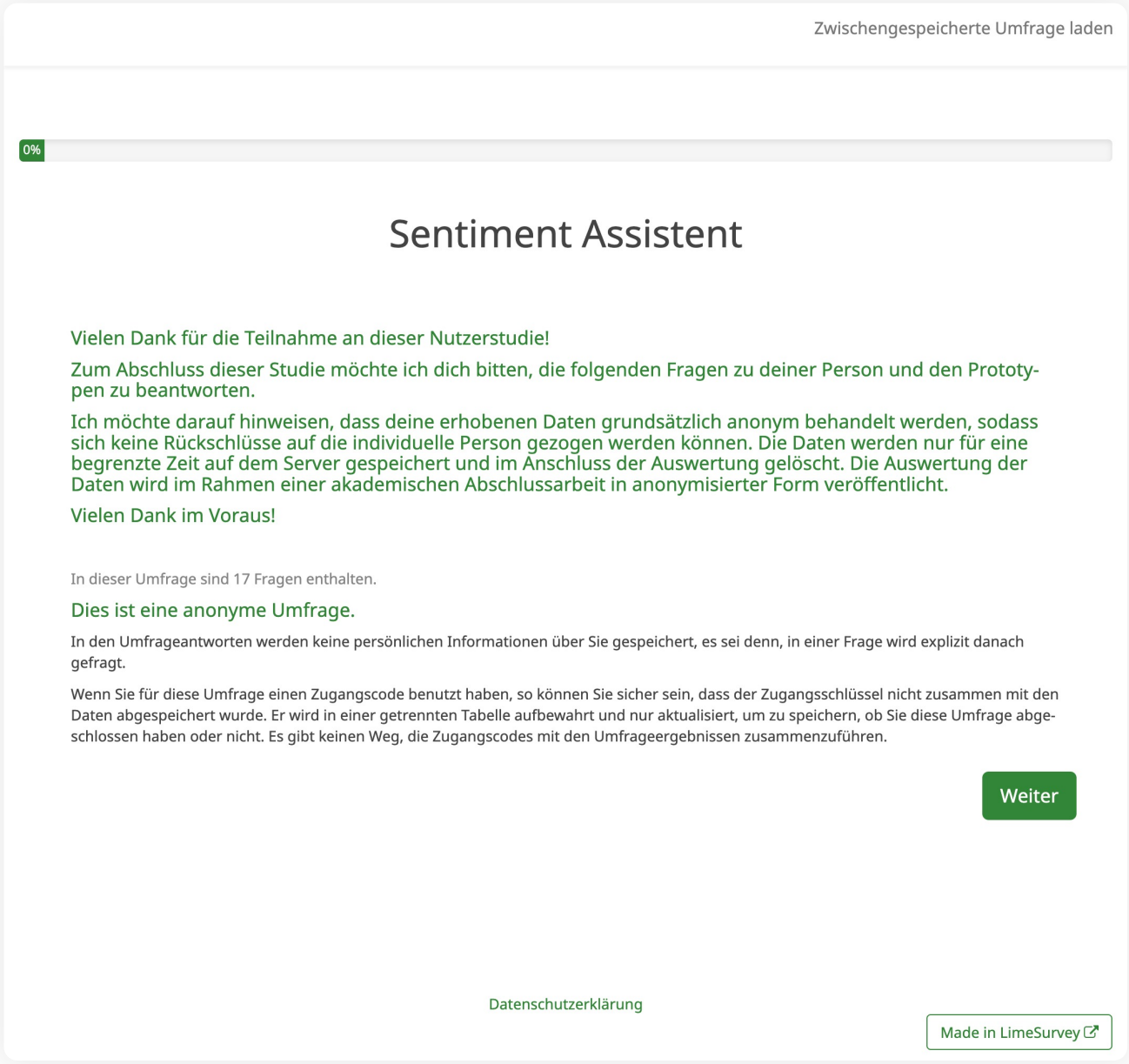


Abbildung C.1: Fragebogen Willkommensseite

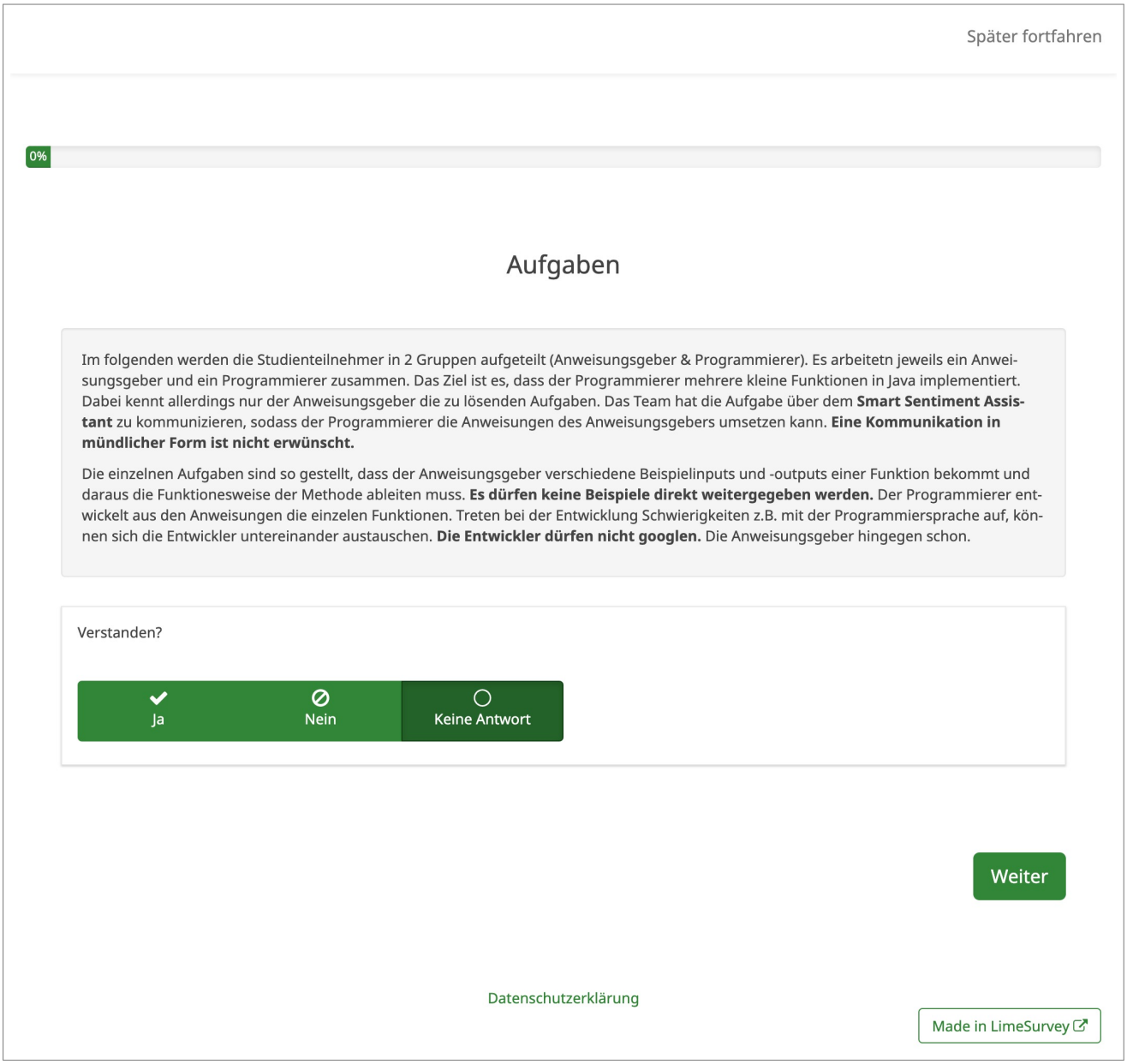
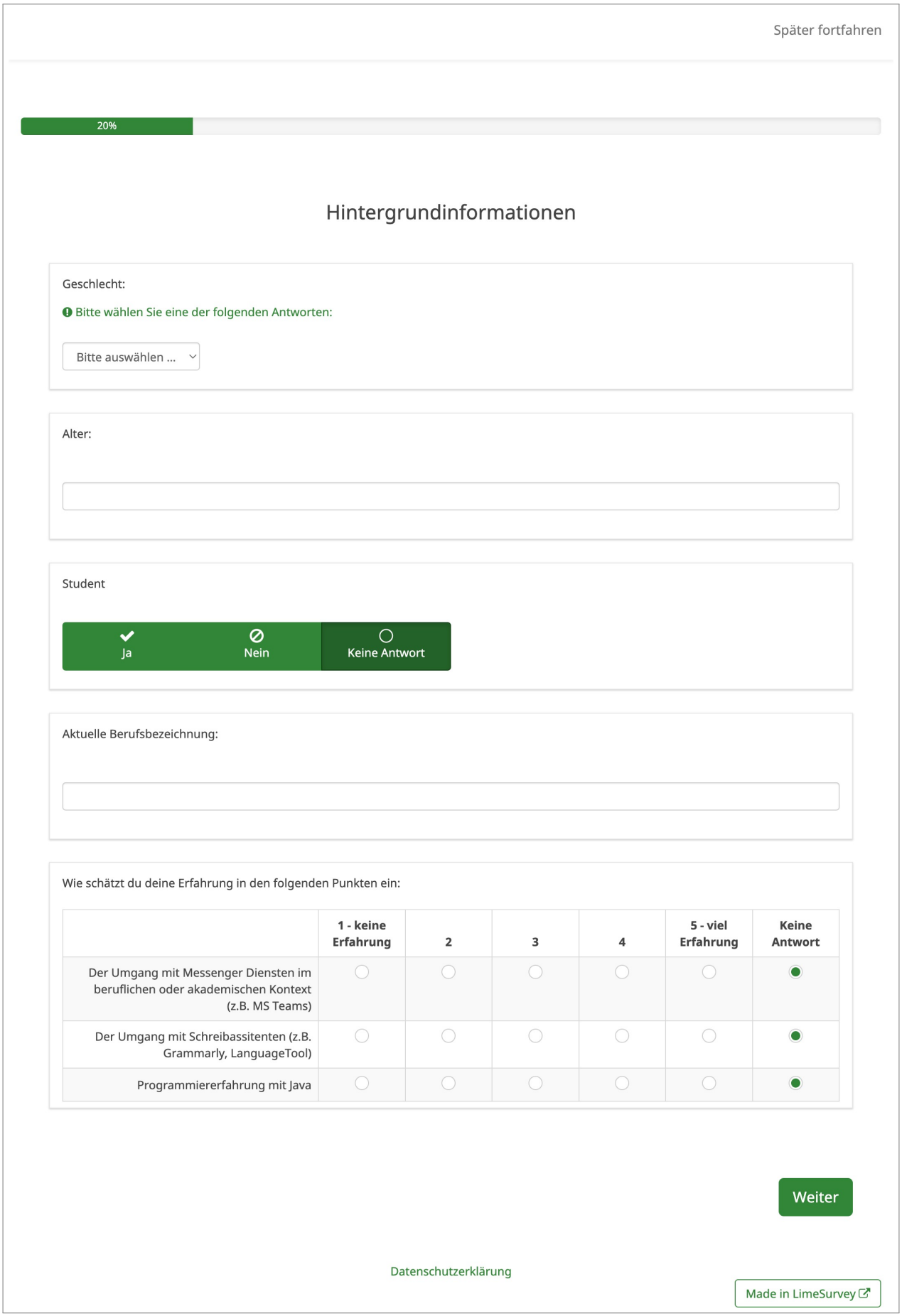


Abbildung C.2: Fragebogen Aufgabenstellung



Später fortfahren

20%

Hintergrundinformationen

Geschlecht:

Bitte wählen Sie eine der folgenden Antworten:

Bitte auswählen ...

Alter:

Student

Ja | Nein | Keine Antwort

Aktuelle Berufsbezeichnung:

Wie schätzt du deine Erfahrung in den folgenden Punkten ein:

	1 - keine Erfahrung	2	3	4	5 - viel Erfahrung	Keine Antwort
Der Umgang mit Messenger Diensten im beruflichen oder akademischen Kontext (z.B. MS Teams)	○	○	○	○	○	●
Der Umgang mit Schreibassitenten (z.B. Grammarly, LanguageTool)	○	○	○	○	○	●
Programmiererfahrung mit Java	○	○	○	○	○	●

Weiter

Datenschutzerklärung

Made in LimeSurvey

40%

Evaluation Prototyp

	1 - Stimme gar nicht zu	2	3	4 - Stimme völlig zu	Keine Antwort
Ich habe häufig auf die Einfärbung des Textfeldes geachtet (grüner Hintergrun & rote Textmarkierung).	○	○	○	○	●
Ich habe häufig den rot markierten Satz angeklickt, um die Umformulierung zu sehen.	○	○	○	○	●
Ich habe häufig den umformulierten Satz statt meinen eigenen verwendet.	○	○	○	○	●
Ich habe häufig die Rechtschreibüberprüfung genutzt (rot unterkringelter Text).	○	○	○	○	●

	1 - Stimme gar nicht zu	2	3	4 - Stimme völlig zu	Keine Antwort
Ich empfand die farblichen Einblendungen als störend.	○	○	○	○	●
Ich empfand rot markierten Sätze tatsächlich im Bezug auf Stimmung als Negativ.	○	○	○	○	●
Ich empfand die Berechnung der Umformulierungen als zu lang.	○	○	○	○	●
Ich empfand die Umformulierungen vom Inhalt als passend.	○	○	○	○	●
Ich empfand die Umformulierungen im Bezug auf die Stimmung positiver als der Ausgangssatz.	○	○	○	○	●

Warum fandest du die farblichen Markierungen störend bzw. nicht störend?

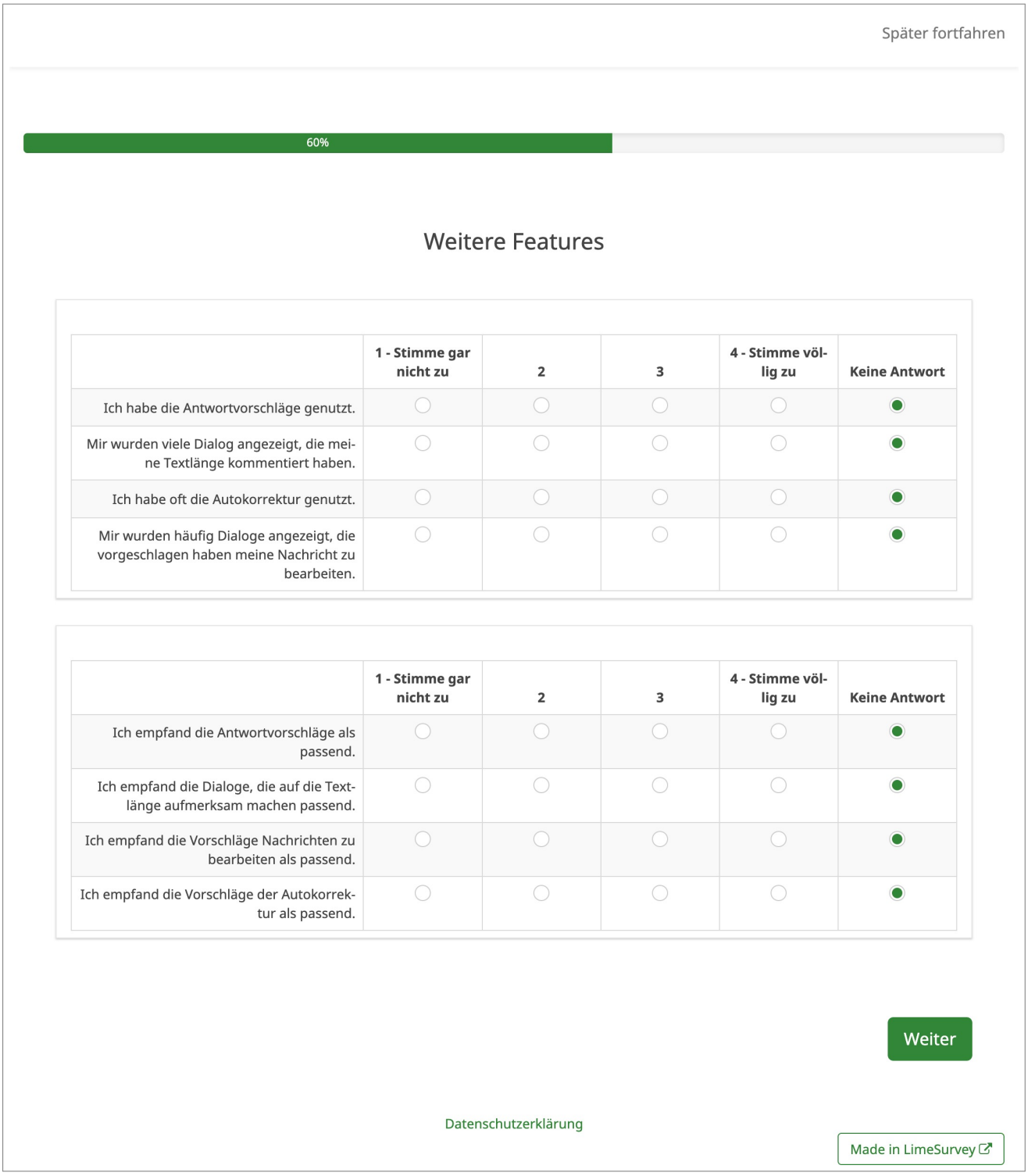
Was hätte ich mir für die Anzahl der roten (negativen) Nachrichten gewünscht? Wurden zu viele oder zu wenige Nachrichten negativ eingeordnet?

Was war der Grund wenn du Umformulierungen nicht genutzt hast?

Was war der Grund wenn du rot markierte Sätze nicht angeklickt hast?

	1 - Stimme gar nicht zu	2	3	4 - Stimme völlig zu	Keine Antwort
Ich empfand die Nutzung des Prototypen insgesamt als Sinnvoll.	○	○	○	○	●
Ich könnte mir einen Einsatz in einem Entwicklungsteam vorstellen.	○	○	○	○	●
Ich kann mir vorstellen, dass sich die Gesamtstimmung durch den Prototypen verbessern kann.	○	○	○	○	●

Folgende Sachen würde ich mir für einen Einsatz in einem Entwicklungsteam wünschen?



Später fortfahren

60%

Weitere Features

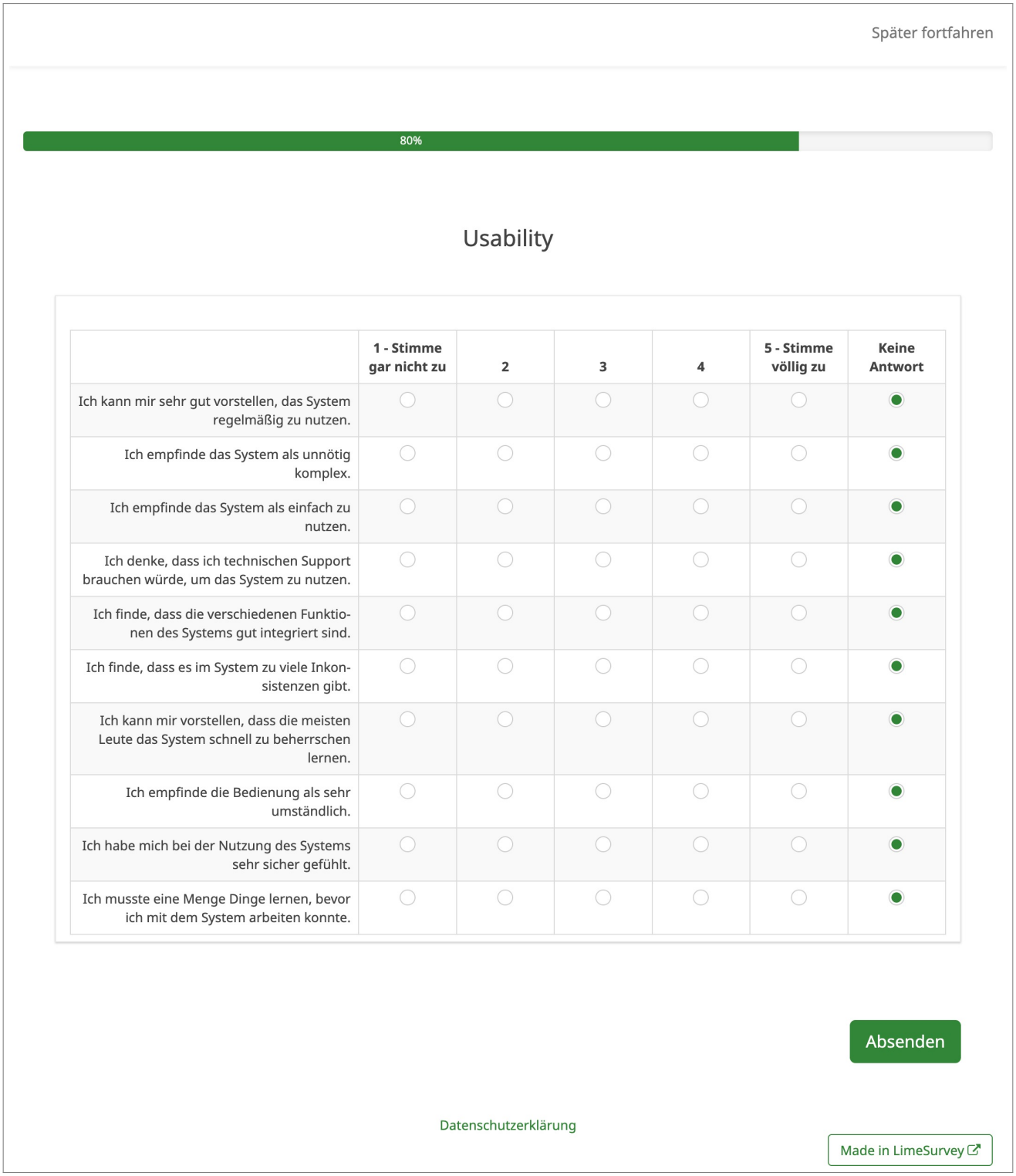
	1 - Stimme gar nicht zu	2	3	4 - Stimme völlig zu	Keine Antwort
Ich habe die Antwortvorschläge genutzt.	○	○	○	○	●
Mir wurden viele Dialog angezeigt, die meine Textlänge kommentiert haben.	○	○	○	○	●
Ich habe oft die Autokorrektur genutzt.	○	○	○	○	●
Mir wurden häufig Dialoge angezeigt, die vorgeschlagen haben meine Nachricht zu bearbeiten.	○	○	○	○	●

	1 - Stimme gar nicht zu	2	3	4 - Stimme völlig zu	Keine Antwort
Ich empfand die Antwortvorschläge als passend.	○	○	○	○	●
Ich empfand die Dialoge, die auf die Textlänge aufmerksam machen passend.	○	○	○	○	●
Ich empfand die Vorschläge Nachrichten zu bearbeiten als passend.	○	○	○	○	●
Ich empfand die Vorschläge der Autokorrektur als passend.	○	○	○	○	●

Weiter

Datenschutzerklärung

Made in LimeSurvey



Später fortfahren

80%

Usability

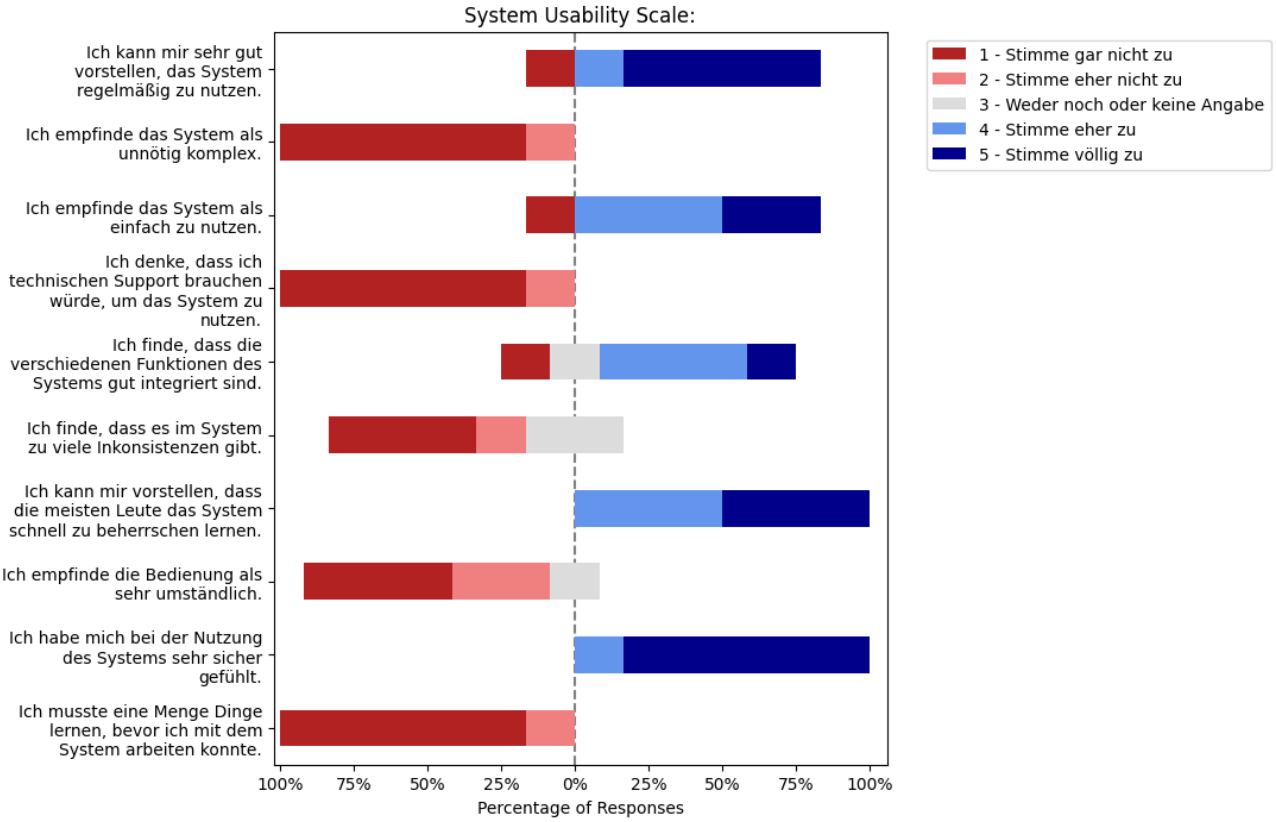
	1 - Stimme gar nicht zu	2	3	4	5 - Stimme völlig zu	Keine Antwort
Ich kann mir sehr gut vorstellen, das System regelmäßig zu nutzen.	○	○	○	○	○	●
Ich empfinde das System als unnötig komplex.	○	○	○	○	○	●
Ich empfinde das System als einfach zu nutzen.	○	○	○	○	○	●
Ich denke, dass ich technischen Support brauchen würde, um das System zu nutzen.	○	○	○	○	○	●
Ich finde, dass die verschiedenen Funktionen des Systems gut integriert sind.	○	○	○	○	○	●
Ich finde, dass es im System zu viele Inkonsistenzen gibt.	○	○	○	○	○	●
Ich kann mir vorstellen, dass die meisten Leute das System schnell zu beherrschen lernen.	○	○	○	○	○	●
Ich empfinde die Bedienung als sehr umständlich.	○	○	○	○	○	●
Ich habe mich bei der Nutzung des Systems sehr sicher gefühlt.	○	○	○	○	○	●
Ich musste eine Menge Dinge lernen, bevor ich mit dem System arbeiten konnte.	○	○	○	○	○	●

Absenden

Datenschutzerklärung

Made in LimeSurvey

C.2 System Usability Scale



C.3 Aufgabenstellung

Anweisungsgeber 1

Aufgabe 1a) // für UserTwo

In: [1, 2, 3, 5]	Out: [8, 13]
In: [21, 34, 55]	Out: [89, 144]
In: [4, 8]	Out: [12, 20]

Aufgabe 2a) // für UserTwo

In: 453	Out: [„drei“, „vier“, „fünf“]
In: 682	Out: [„zwei“, „sechs“, „acht“]
In: 6785	Out: [„fünf“, „sechs“, „sieben“, „acht“]

Aufgabe 3c) // für UserSix

In: „Hallo“	Out: „OLLAh“
In: „Computer“	Out: „RETUPMOc“
In: „InforMatik“	Out: „KITAmROFNi“

Aufgabe 4c) // für UserSix

In: [„Haus“, „Welt“, „Boot“, „Haus“]	Out: [„Haus“, „Haus“]
In: [„Katze“, „Hund“, „Katze“, „Maus“]	Out: [„Katze“, „Katze“]
In: [„A“, „B“, „B“]	Out: [„B“, „B“]

Aufgabe 5b) // für UserFour

In: [4, 78, 23, 74, 0, 3]	Out: [78, 0, 74, 3, 23, 4]
In: [56, 849, 4, 55, 98, 23]	Out: [849, 4, 98, 23, 56, 55]
In: [-5, 14, 1000, 0]	Out: [1000, -5, 14, 0]

Aufgabe 6b) // für UserFour

In: [„4“, „7“, „98“, „38“], 4	Out: 0
In: [„7“, „87“, „28“, „9“], 28	Out: 2
In: [„90“, „66“, „1267“], 22	Out: -1
In: [„12“, „-123“], -123	Out: 1

Anweisungsgeber 3

Aufgabe 1b) // für UserFour

In: 1, 2, 2, 4	Out: 8, 32
In: 3, 2, 6	Out: 12, 72
In: 4, 8	Out: [32, 256]

Aufgabe 2b) // für UserFour

In: 453	Out: [„Drei“, „Vier“, „Fünf“]
In: 682	Out: [„Zwei“, „Sechs“, „Acht“]
In: 6785	Out: [„Fünf“, „Sechs“, „Sieben“, „Acht“]

Aufgabe 3a) // für UserTwo

In: „Hallo“	Out: „ollah“
In: „Masterarbeit“	Out: „tiebraretsam“
In: „Informatik“	Out: „kitamrofni“

Aufgabe 4a) // für UserTwo

In: [„Haus“, „Welt“, „Boot“, „Haus“]	Out: [„Haus“, „Welt“, „Boot“]
In: [„Katze“, „Hund“, „Katze“, „Maus“]	Out: [„Katze“, „Hund“, „Maus“]
In: [„A“, „B“, „B“]	Out: [„A“, „B“]

Aufgabe 5c) // für UserSix

In: [4, 78, 23, 74, 0, 3]	Out: [-78, 0, -74, 3, -23, 4]
In: [56, 849, 4, 55, 98, 23]	Out: [-849, 4, -98, 23, -56, 55]
In: [-5, 14, 1000, 0]	Out: [-1000, -5, -14, 0]

Aufgabe 6c) // für UserSix

In: [„4“, „7“, „98“, „38“], 4	Out: 0
In: [„7“, „87“, „28“, „9“], 28	Out: 2
In: [„90“, „66“, „1267“], 22	Out: -1
In: [„12“, -123], -123	Out: 1

Anweisungsgeber 5

Aufgabe 1c) // für UserSix

In: 54, 20, 24	Out: -4, 28
In: 10, 3, 7	Out: -4, 11
In: 4, 8	Out: [-4, 12]

Aufgabe 2c) // für UserSix

In: 453	Out: [„Fünf“, „Vier“, „Drei“]
In: 682	Out: [„Acht“, „Sechs“, „Zwei“]
In: 6785	Out: [„Acht“, „Sieben“, „Sechs“, „Fünf“]

Aufgabe 3b) // für UserFour

In: „Hallo“	Out: „ollaH“
In: „Computer“	Out: „retupmoC“
In: „InforMatik“	Out: „kitaMrofnI“

Aufgabe 4b) // für UserFour

In: [„Haus“, „Welt“, „Boot“, „Haus“]	Out: [„Welt“, „Boot“]
In: [„Katze“, „Hund“, „Katze“, „Maus“]	Out: [„Hund“, „Maus“]
In: [„A“, „B“, „B“]	Out: [„A“]

Aufgabe 5a) // für UserTwo

In: [4, 78, 23, 74, 0, 3]	Out: [0, 78, 3, 74, 4, 23]
In: [56, 849, 4, 55, 98, 23]	Out: [4, 849, 23, 98, 55, 56]
In: [-5, 14, 1000, 0]	Out: [-5, 1000, 0, 14]

Aufgabe 6a) // für UserTwo

In: [„4“, „7“, „98“, „38“], 4	Out: 1
In: [„7“, „87“, „28“, „9“], 28	Out: 3
In: [„90“, „66“, „1267“], 22	Out: 0
In: [„12“, „-123“], -123	Out: 2

Abbildungsverzeichnis

Literaturverzeichnis

[1] J. Ahn, Y. Lee, J. Ahn, and J. Ko. Server load and network-aware adaptive deep learning inference offloading for edge platforms. *Internet of Things*, 21:100644, 2023.

[2] S. W. Ambler. Agile software development at scale. In *IFIP Central and East European Conference on Software Engineering Techniques*, pages 1–12. Springer, 2007.

[3] C. Anderson. Docker [software engineering]. *IEEE Software*, 32(3):102–c3, 2015.

[4] A. Anwar, X. Li, Y. Yang, R. Dong, and T. Osman. Constructing uyghur named entity recognition system using neural machine translation tag projection. In *Chinese Computational Linguistics*, pages 247–260, Cham, 2020. Springer International Publishing.

[5] S. Baccianella, A. Esuli, F. Sebastiani, et al. Sentiwordnet 3.0: an enhanced lexical resource for sentiment analysis and opinion mining. In *Lrec*, volume 10, pages 2200–2204, 2010.

[6] A. Bangor, P. Kortum, and J. Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

[7] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[8] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli. Sentiment polarity detection for software development. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, page 128, New York, NY, USA, 2018. Association for Computing Machinery.

[9] C. Costa Silva, F. Gilson, and M. Galster. Comparison framework for team-based communication channels. In X. Franch, T. Männistö, and S. Martínez-Fernández, editors, *Product-Focused Software Process Improvement*, pages 315–322, Cham, 2019. Springer International Publishing.

[10] R. Dale and J. Viethen. The automated writing assistance landscape in 2021. *Natural Language Engineering*, 27(4):511–518, 2021.

[11] C. Federmann, O. Elachqar, and C. Quirk. Multilingual whispers: Generating paraphrases with translation. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 17–26, Hong Kong, China, Nov. 2019. Association for Computational Linguistics.

[12] D. Ford, M.-A. Storey, T. Zimmermann, C. Bird, S. Jaffe, C. Maddila, J. L. Butler, B. Houck, and N. Nagappan. A tale of two cities: Software developers working from home during the covid-19 pandemic. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(2):1–37, 2021.

[13] O. Guhr, A.-K. Schumann, F. Bahrmann, and H. J. Böhme. Training a broad-coverage german sentiment classification model for dialog systems. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 1627–1632, 2020.

[14] E. Guzman and B. Bruegge. Towards emotional awareness in software development teams. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, page 671–674, New York, NY, USA, 2013. Association for Computing Machinery.

[15] K. Hornik and D. Murdoch. Watch your spelling! *The R Journal*, 3(2):22–28, 2011.

[16] M. R. Islam and M. F. Zibran. Sentistrength-se: Exploiting domain specificity for improved sentiment analysis in software engineering text. *Journal of Systems and Software*, 145:125–146, 2018.

[17] M. Jiménez, M. Piattini, and A. Vizcaíno. Challenges and improvements in distributed software development: A systematic review. *Advances in Software Engineering*, 2009, 2009.

[18] F. Jurado and P. Rodriguez. Sentiment analysis in monitoring software development processes: An exploratory case study on github's project issues. *Journal of Systems and Software*, 104:82–89, 2015.

[19] Y. Kaya, S. Hong, and T. Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pages 3301–3310. PMLR, 2019.

[20] S. A. Kirkpatrick. *Build a better vision statement: Extending research with practical advice*. Rowman & Littlefield, 2016.

[21] J. Kollmann, H. Sharp, and A. Blandford. The importance of identity and vision to user experience designers on agile projects. In *2009 Agile Conference*, pages 11–18. IEEE, 2009.

[22] J. R. Lewis. The system usability scale: Past, present, and future. *International Journal of Human–Computer Interaction*, 34(7):577–590, 2018.

[23] R. Mitton. Fifty years of spellchecking. *Writing Systems Research*, 2(1):1–7, 2010.

[24] E. Murphy-Hill and G. C. Murphy. Recommendation delivery: Getting the user interface just right. In *Recommendation systems in software engineering*, pages 223–242. Springer, 2013.

[25] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, mar 2001.

[26] R. Neumann and J. Kozlik. Emotionsausdruck und emotionale ansteckung. *Kommunikation, Interaktion und soziale Gruppenprozesse. Enzyklopädie der Psychologie*, pages 163–190, 2017.

[27] J. NIELSEN. Chapter 5 - usability heuristics. In *Usability Engineering*, pages 115–163. 1993.

[28] N. Novielli and A. Serebrenik. Sentiment and emotion in software engineering. *IEEE Software*, 36(5):6–23, 2019.

[29] M. Obaidi and J. Klünder. Development and application of sentiment analysis tools in software engineering: A systematic literature review. *Evaluation and Assessment in Software Engineering*, pages 80–89, 2021.

[30] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 281–290, 2015.

[31] R. Quirk and D. Crystal. *A comprehensive grammar of the English language*. Pearson Education India, 2010.

[32] R. Sandoval-Almazan and D. Valle-Cruz. Facebook impact and sentiment analysis on political campaigns. In *Proceedings of the 19th Annual International Conference on Digital Government Research: Governance in the Data Age*, dg.o '18, New York, NY, USA, 2018. Association for Computing Machinery.

[33] L. Schroth, M. Obaidi, A. Specht, and J. Klünder. On the potentials of realtime sentiment analysis on text-based communication in software projects. In R. Bernhaupt, C. Ardito, and S. Sauer, editors, *Human-Centered Software Engineering*, pages 90–109, Cham, 2022. Springer International Publishing.

[34] U. Sidarenka. PotTS: The Potsdam Twitter sentiment corpus. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1133–1141, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).

[35] L. Solbakken. Machine-learned model serving at scale, 2022. https://towardsdatascience.com/machine-learned-model-serving-at-scale-86f6220132c8, Aufgerufen am 06.07.2023.

[36] M. Thelwall. The heart and soul of the web? sentiment strength detection in the social web with sentistrength. *Cyberemotions: Collective emotions in cyberspace*, pages 119–134, 2017.

[37] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: open and efficient foundation language models, 2023. *URL https://arxiv.org/abs/2302.13971*.

[38] E. Van Kelle, J. Visser, A. Plaat, and P. van der Wijst. An empirical study into social success factors for agile software development. In *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 77–80. IEEE, 2015.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[40] F. Witte. *Goal Question Metric*, pages 145–150. Springer Fachmedien Wiesbaden, Wiesbaden, 2018.

[41] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[42] M. Wojatzki, E. Ruppert, S. Holschneider, T. Zesch, and C. Biemann. Germeval 2017: Shared task on aspect-based sentiment in social media customer feedback. *Proceedings of the GermEval*, pages 1–12, 2017.

[43] M. R. Wrobel. Emotions in the software development process. In *2013 6th International Conference on Human System Interactions (HSI)*, pages 518–523. IEEE, 2013.

[44] G.-t. Wu. Research on management over-optimistic measurement: Based on text sentiment analysis. In *European Journal of Business and Management*, 2019.

[45] F. Yvon. Transformers in natural language processing. In *ECCAI Advanced Course on Artificial Intelligence*, pages 81–105. Springer, 2021.

[46] R. Zhang, J. Han, A. Zhou, X. Hu, S. Yan, P. Lu, H. Li, P. Gao, and Y. Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023.