Entwicklung eines mehrbenutzerfähigen Quelltexteditors für SEE

Moritz Blecker

Bachelorarbeit

Fachbereich 03
Universität Bremen

1. Gutachter: Prof. Dr. Rainer Koschke
2. Gutachterin: Dr. Hui Shi

Abgabedatum: 31.01.2022

Abstract

In dieser Bachelorarbeit wird untersucht, ob die Ergebnisse von Aufgaben innerhalb von *SEE* welche von mehreren Benutzern gleichzeitig bearbeitet werden, effektiver mithilfe eines in *SEE* integrierten mehrbenutzerfähigen Quelltexteditors gelöst werden können oder ob ein einfacher Windows-Editor ausreichend ist.
Dazu wird zunächst beschrieben, wie ein solcher Editor in *SEE* implementiert werden kann. Im Anschluss wird diese Implementierung dann getestet und in Hinblick auf die oben genannte Fragestellung ausgewertet. Diese Evaluation wurde als vergleichende Benutzerstudie zwischen den beiden genannten Editoren durchgeführt. Insgesamt wurde in der Studie eine Teilnehmerzahl von $n = 24$ erreicht. Als Ergebnis dieser Evaluation wurde herausgefunden, dass zwar innerhalb der Studie die Aufgaben schneller und korrekter mit dem SEE-Editor gelöst werden konnten, dies war allerdings kein statistisch signifikantes Ergebnis. Hingegen konnte jedoch gezeigt werden, dass es einen signifikanten Unterschied in Bezug auf die Benutzbarkeit macht, ob der SEE-Editor oder der Windows-Editor benutzt wurde. Der SEE-Editor wurde als angenehmeres Mittel zur Lösung der Aufgabe empfunden.
Es wurde dennoch festgestellt, dass im Rahmen dieser Arbeit kein fehlerfreier Editor erstellt werden konnte. So kann es in bestimmten Fällen passieren, dass die Synchronisierung des Inhaltes nicht korrekt funktioniert. Des Weiteren gibt es noch Optimierungspotenzial in Bezug auf die Benutzbarkeit. So funktioniert insbesondere das Scrollen in einer geöffneten Datei nicht optimal.

Erklärung

Ich versichere, die Bachelorarbeit ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Bremen, den 31.1.2022

. . . . . .
(Moritz Blecker)

Gender-Hinweis

Aus Gründen der besseren Lesbarkeit wird bei Personenbezeichnungen und personenbezogenen Hauptwörtern in dieser Bachelorarbeit die männliche Form verwendet. Entsprechende Begriffe gelten im Sinne der Gleichbehandlung grundsätzlich für alle Geschlechter und beinhalten keine Wertung.[1]

[1] Gender-Hinweis in leicht angepasster Form. Übernommen von: https://www.randstad.de/ueber-randstad/gender-hinweis/

Danksagung

Zuallererst möchte ich mich bei Prof. Dr. Rainer Koschke bedanken, als Betreuer war er an jedem Wochentag verfügbar, um auf Fragen zur Technik, aber auch zu Formalien rund um die Bachelorarbeit zu antworten. Diese oft kurzfristigen Antworten haben den Ablauf der Arbeit enorm erleichtert.
Zusätzlich gilt ein besonderer Dank an Falko Galperin, dieser konnte bei zahlreichen Fragen zu Problemen mit Unity, C# und auch Latex mit hilfreichen Tipps und Tricks weiterhelfen. Zudem möchte ich mich bei ihm zusätzlich dafür bedanken, dass er einen Teil dieser Arbeit Korrektur gelesen hat.
Ein großer Dank geht auch an Thore Frenzel, dieser hat einen Großteil dieser Arbeit Korrektur gelesen und sich nicht von dem großen Zeitaufwand hierfür abschrecken lassen.
Zuletzt möchte ich mich noch bei allen Teilnehmern meiner Evaluation bedanken. Ohne eure rege Teilnahme wäre die Evaluation nicht möglich gewesen.
An dieser Stelle ist auch ein Dank an Anna Blecker zurichten. Durch Ihre Hilfe konnte die manuelle Auswertung der Korrektheit der Lösungen mehrfach überprüft werden.

INHALTSVERZEICHNIS

Kapitel 1 EINLEITUNG

In dieser Arbeit wird die Erstellung eines mehrbenutzerfähigen Quelltexteditors für die Software-SEE und dessen Leistungsfähigkeit beschrieben und evaluiert. Im folgenden Kapitel wird zunächst eine Einführung in das Thema gegeben und darauf aufbauend werden die Ziele dieser Arbeit vorgestellt.

1.1 Probleme in der Softwareentwicklung

Seit einigen Jahren, zunehmend seit Beginn der Covid-19-Pandemie, gibt es weltweit den Trend zum „Homeoffice“ oder auch „Distributed Development“. Dadurch, dass die einzelnen Entwickler sich immer weniger in gleichen Räumen befinden und die räumliche Distanz zwischen ihnen durch zuvor angesprochenen Entwicklungen immer größer wird, muss die Kommunikation immer mehr durch Technik unterstützt werden. Gerade im Bereich der Softwareentwicklung existieren hier noch große Probleme.
Gründe hierfür sind einerseits, dass Quellcode als Kommunikationsgrundlage häufig zu detailreich ist, um ein gutes Verständnis von der gesamten Software zu erwerben, andererseits aber auch, weil häufig nur eine Person aus der Gruppe aktiv Änderungen vornehmen kann. Der Einsatz von Videochat-Systemen wie Zoom oder Microsoft-Teams und deren Screen-Sharing-Funktionen sorgt dafür, dass der Austausch auf das Zuschauen bei der Arbeit eines anderen beschränkt ist. Dadurch kann zwar jeder Teilnehmer sehen, was vor sich geht, hat jedoch keinen direkten Einfluss auf den Inhalt, sei es, einen bestimmten Abschnitt noch einmal zu lesen oder aber auch diesen zu bearbeiten. Zusätzlich wird Software in ihrer Funktionalität, Struktur und ihrem Umfang zunehmend komplexer. Dadurch wird es immer schwieriger für Entwickler, sie in Gänze zu verstehen und vor allem sich in bereits bestehende Software einzuarbeiten.
Sogar der Austausch zwischen z. B. Arbeitskollegen über eigens entwickelte Software stellt viele Entwickler vor Probleme, da häufig auf unterschiedlichen Endgeräten gearbeitet wird.

1.2 Software Engeneering Experience

Aus diesen Gründen wurde von der AG Softwaretechnik der Universität Bremen die Software *Software Engineering Experience* (SEE) ins Leben gerufen. Das Ziel von SEE ist es, die Zusammenarbeit von Entwicklern in verteilten Teams und das Verständnis dieser vom eigenen oder auch fremden Code zu verbessern. Als Lösungsansatz benutzt SEE ein Konzept zur Visualisierung von Code. So kann z. B. eine komplette Software innerhalb einer 3D-Applikation vi-

SEE: Eine Software zur Visualisierung von Quellcode.

sualisiert werden. Dazu wird die Software mit all ihren Komponenten auf eine Stadt, eine sogenannte Software-City, abgebildet.

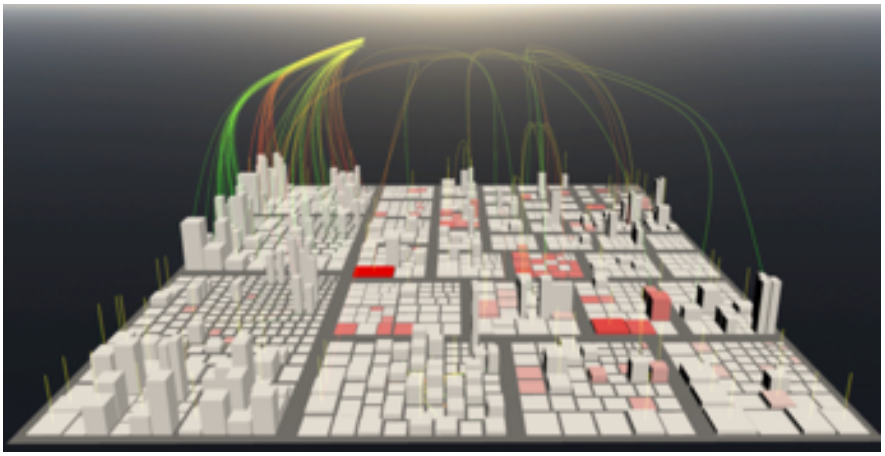

Abbildung 1: Eine Software als Stadt in SEE.

Ein Beispiel für die zuvor angesprochene Visualisierung ist in Abbildung Abbildung 1 zu sehen. In dieser Abbildung stehen die Gebäude für Klassen und die Stadtviertel, in denen Sie stehen, für das Paket, zu dem sie gehören. Durch die bunt gefärbten Linien und Verbindungen zwischen den Gebäuden werden Verbindungen zwischen den Klassen, wie ein Aufruf einer anderen Klasse, abgebildet.
Es lässt sich konfigurieren, dass die einzelnen Komponenten der Stadt für andere Werte in der Software als Klassen stehen, zum Beispiel könnten die Gebäude auch einzelne Methoden und die Stadtviertel die Klassen dazu darstellen.
Die Farbe, Höhe, Breite und Tiefe der Gebäude bilden verschiedenen Softwaremetriken ab. Sie können zum Beispiel für die *McCabe Komplexität*, die *Anzahl an Codezeilen* (LOC), die Anzahl an aufgerufenen Klassen oder auch die Anzahl an aufrufenden Klassen stehen. Diese Metriken können frei gewählt werden. Grundsätzlich kann hierdurch alles dargestellt werden, was sich in Form von Zahlen auf Skalen erfassen lässt. [Kos20]
In einer Software-City ist es auch möglich, sich den zugrunde liegenden Quelltext anzuschauen. Dazu wird dann ein sogenanntes Code-Window im Sichtfeld des Benutzers eingeblendet, in welchem man die komplette Klasse, die zuvor ausgewählt wurde, einsehen kann (siehe Abbildung 2). Innerhalb des Code-Windows, wird, wie bei gängigen IDE's und Code-Editoren, *Syntax-Highlighting* genutzt. Zusätzlich kann dieses Code-Window auch im Mehrbenutzer-Betrieb verwendet werden, hier wird dann der betrachtete Aus-

McCabe Komplexität: Eine Metrik, um die Komplexität von Quellcodes anzugeben.

LOC: Lines of Code, eine Metrik, die die Zeilenanzahl einer Klasse angibt.

Syntax-Highlighting: Die farbliche Hervorhebung bestimmter Schlüsselwörter einer Programmiersprache im Quellcode.

schnitt des Codes zwischen den Nutzern synchronisiert, damit jeder Nutzer dieselbe Stelle des Codes betrachten kann. Diese Code-Windows sollen nun im Rahmen dieser Arbeit erweitert werden, sodass ein voll funktionaler, synchronisierter Code-Editor innerhalb von SEE integriert wird. Anschließend soll dann noch überprüft werden, ob dieser für die Benutzer einen Mehrwert bietet.

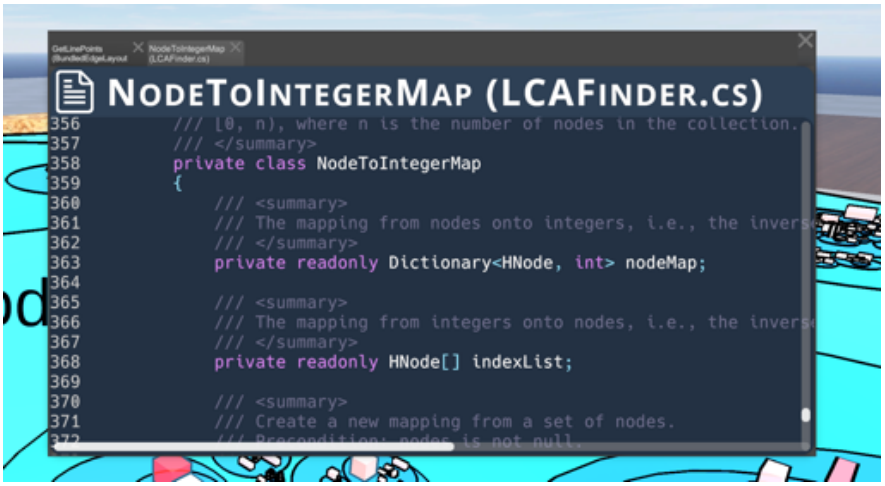

Abbildung 2: Ein Code-Window.

Kapitel 2 ZIEL DER ARBEIT

Aus dieser Erweiterung leitet sich die Forschungsfrage für die Arbeit wie folgt ab: Wie unterscheidet sich die Effektivität kollaborativen Arbeitens bei einem in SEE eingebetteten, mehrbenutzerfähigen Editor von der Effektivität der Nutzung eines normalen Texteditors? Um diese Frage zu klären, musste zunächst das vorhandene Code-Window zu einem kollaborativen Editor umgearbeitet werden. Dazu sollen folgende Punkte erreicht werden:

- Das Code-Window soll um die Möglichkeit der Texteingabe erweitern.
- Die verschiedenen Versionen der Texte unterschiedlicher Nutzer sollen, in nahezu Echtzeit, synchronisiert und angepasst werden.

Optional soll der Editor noch folgende Funktionen enthalten.

- Rückgängig machen von Änderungen, allgemein auch als Undo und Redo bezeichnet.
- Speichern von Änderungen
- Anzeigen, an welcher Stelle im Text andere Nutzer arbeiten.

Nachdem der Editor implementiert wurde, soll dieser dann im Bezug zur Forschungsfrage evaluiert werden. Dazu soll eine vergleichende Benutzerstudie durchgeführt werden, welche die folgenden drei Leitfragen beantwortet.

- Können Benutzer mithilfe des Editors kollaborative Aufgaben in SEE schneller lösen, als mit einem nicht kollaborativen Editor
- Sind die Ergebnisse kollaborativer Aufgaben in SEE bei dem integrierten Editor korrekter, als bei einem anderen nicht kollaborativen Editor.
- Ist es für die Nutzer angenehmer kollaborative Aufgaben in SEE mit dem Editor zu lösen im Vergleich zu einem nicht kollaborativen Editor.

Kapitel 3 GRUNDLAGEN

Nachdem nun die Ziele dieser Arbeit bekannt sind, werden im folgenden Kapitel, zunächst einige, zum besseren Verständnis der folgenden Kapitel notwendigen, Grundlagen erklärt.

3.1 SEE

Die Anwendung SEE bietet verschiedene Optionen zur Interaktion, die in vier Kernaspekte gruppiert werden können:

Abbildung 3: Ein Blick auf SEE.

- Im linken, vorderen Bereich der Abbildung ist die Visualisierung einer Softwarearchitektur zu sehen. Die Architektur, also der Soll-Zustand einer Software sollte im Optimalfall regelmäßig mit dem Ist-Zustand der Software verglichen werden. Häufig wird dies aber der Integration neuer Funktionen hinten angestellt und somit zunehmend vernachlässigt — die Software erodiert. In SEE ist es möglich, diesen Bauplan direkt mit der Implementierung selbst zu vergleichen und Unterschiede und Gemeinsamkeiten zwischen diesen beiden farblich hervorzuheben. Dieser Bereich von SEE wird als **Architekturvergleich** bezeichnet.

- Die Software-City vorne rechts zeigt die Darstellung der **Implementierung** einer Software. Das Design der Darstellung ist, wie oben beschrieben, flexibel — so stehen neben dem gezeigten Layout auch noch andere Layouts zur Verfügung. Jedes Layout hat in seiner Erscheinung andere

Schwerpunkte. Die Gemeinsamkeit aller Layouts ist allerdings, dass die Blöcke als Häuser einer Software-City erhalten bleiben. Lediglich das Design der Stadtviertel wird in den verschiedenen Formaten verändert.

- Auf dem Tisch hinten rechts ist eine **Softwareevolution** abgebildet, also wie sich eine Software über die Zeit der Entwicklung verändert hat. Hier können auch wieder Softwaremetriken über Farbe, Höhe, Tiefe, Breite usw. abgebildet werden. Zusätzlich wird der Nutzer hier noch über sogenannte *Beams* über Veränderungen zwischen zwei Versionen der Software informiert, so stehen die grünen Beams zum Beispiel für neu hinzugefügte Klassen.

Beam: Senkrecht nach oben gerichtet Lichtstrahlen zur Markierung einer bestimmten Stelle.

- Im linken, hinteren Bereich der Abbildung wird die Funktion des **Debuggens** gezeigt. Es wird ein zur Laufzeit aufgenommener Ausschnitt aus der Software wiedergeben. Hier ist die Blockrepräsentation wieder ähnlich, aber die Kanten stehen hier für explizite Aufrufe zwischen Klassen. Zusätzlich kann man sich hier auch noch den dazu relevanten Quellcode in bereits angesprochenen Code-Windows einblenden lassen.

SEE ist im Mehrbenutzer-Betrieb nutzbar. Im mittleren Bereich der Abbildung sind zwei Personen, also Benutzer der Anwendung, zu erkennen. Die Bewegungen der Nutzer werden durch die Avatare angezeigt. Zusätzlich wird aktuell eine Mundbewegung im Avatar eines Nutzers dargestellt, wenn dieser über den integrierten Sprachchat spricht. In Zukunft soll die Mimik um die des Nutzers noch erweitert werden, sodass Reaktionen anderer Nutzer auf das Präsentierte besser erkennbar sind.

3.2 Die Game Engine Unity

Unity ist eine Plattform, mithilfe derer 2D und 3D-Anwendungen erstellt werden können. Das Game Engine ermöglicht es diese Anwendungen für verschiedene Plattformen zu entwickeln, unter anderem für Desktop Computer und Mobile Endgeräte. Unity bringt einige Funktionen mit, welche das Entwickeln einer solchen Anwendung erleichtern. Es wird unter anderem eine Umgebung bereitgestellt, in welcher auch dreidimensionale Szenen erstellt werden können. Zusätzlich können diese Szenen mithilfe von unter anderem C# Skripten interaktiv gestaltet werden. Für diese C# Skripte bietet Unity selbst auch einige Erweiterungen an, die viele Dinge erleichtern. Die so erstellten Szenen können dann direkt in Unity getestet werden.

GameObject Als GameObject werden Objekte in Unity bezeichnet. Diese sind etwa die Gebäude aus der Software Stadt. Diese GameObjects können im Funktionsumfang erweitert werden. Es können sogenannte Komponenten an diese Objekte gehängt werden. Dies ist unter anderem ein C# Skript. Mithilfe dieser Komponenten können die GameObjects dann zur Laufzeit erstellt oder zerstört werden oder auch einfach nur bewegt oder interaktiv gestaltet werden.

MonoBehavior Ein MonoBehavior ist eine Klasse, durch die C# Skripts erweitert werden können. Diese wird vor allem dann benötigt, wenn ein GameObject durch ein C# Skript gesteuert wird. Diese Klasse enthält insbesondere Methoden, die zu unterschiedlichen Zeitpunkten automatisch aufgerufen werden. Es gibt unter anderem eine Methode, die beim Starten des Skripts aufgerufen wird und eine, die bei jedem Bildwechsel aufgerufen wird, also beispielsweise bei 60 Bildern pro Sekunde dann 60 Mal in der Sekunde aufgerufen wird.
Zusätzlich gibt es noch einige Unity Methoden die innerhalb dieses MonoBehaviors ergänzt werden können.

Kapitel 4 KONZEPTION

Um die festlegten Ziele zu erreichen, müssen nun die genauen Anforderungen an das zu entwickelnde System definiert und Konzepte hierfür entwickelt werden. Das Ziel der folgenden Abschnitte ist es, zu erläutern, wie die Entwicklung des Editors konzeptioniert wurde.

4.1 Ermöglichen von Textbearbeitung

Um einen Texteditor zu erstellen ist es wichtig, dass der Benutzer neue Inhalte in Form von neuen Zeichen und Zeichenketten zum bereits vorhandenen Text hinzufügen oder auch löschen kann. Im folgenden Abschnitt wird beschrieben, welche Grundlagen hierfür schon vorhanden waren und wie sie verändert werden mussten, um einen Editor zu erstellen.

4.1.1 Vorhandene Grundlagen

Wie bereits in der Einleitung und in Abbildung 2 dargestellt, gibt es in SEE schon die Möglichkeit, Quellcode von Klassen und Methoden mithilfe von Code-Windows einzusehen. Dieses Code-Window ermöglicht es, dass der Nutzer durch einen Klick auf ein Element in der Code-City den zugrunde liegenden Quellcode einsehen kann.
Zudem wird das Öffnen von mehr als nur einer Datei in einem Code-Window zur selben Zeit und auch der Wechsel zwischen diesen geöffneten Dateien unterstützt. Zusätzlich werden Zeilennummern innerhalb der Anzeige berechnet und angezeigt. Das eigentliche Code-Window ist, wie von modernen Betriebssystemen bekannt, verschiebbar und in der Größe anpassbar. Des Weiteren existierte auch eine rudimentäre Mehrbenutzerfähigkeit, in Bezug darauf, dass der Nutzer zwischen seiner eigenen Ansicht und der Ansicht eines anderen Benutzers wechseln kann, also dessen momentan betrachteten Code ebenfalls einsehen kann. Bei der Betrachtung der Ansicht eines anderen Benutzers wird dann der gezeigte Codeausschnitt zwischen den Benutzern synchronisiert, sodass alle Nutzer den gleichen Inhalt sehen.

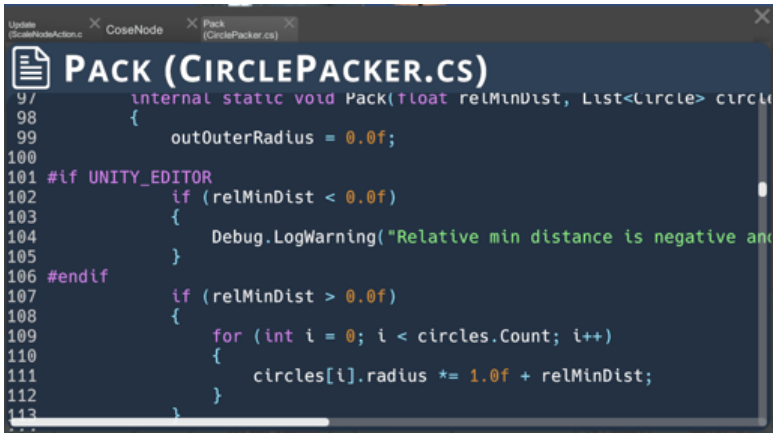

Abbildung 4: Mehrere Tabs in einem Code-Window.

4.1.2 Eingabe von Texten

Diese Grundlage musste dann erweitert werden, sodass zum einen der bestehende Inhalt weiterhin angezeigt wird, er zum anderen aber auch bearbeitet und um weiteren Inhalt ergänzt werden kann. Am effektivsten zur Erreichung dieser Funktionalität eignet sich die Idee, die Anzeige des Textes in ein Eingabefeld zu integrieren, genauer gesagt die Anzeige durch ein Eingabefeld zu ersetzen. In dieses Eingabefeld wird dann beim Öffnen des Code-Windows zunächst der bestehende Inhalt hinein geladen. Dieser kann dann innerhalb des Eingabefeldes beliebig geändert werden.

Da es mit Unity verschiedene Möglichkeiten gibt, Eingabefelder zu erstellen, muss zunächst recherchiert werden, welche sich hierfür am besten eignet. Dabei wurde zunächst das bereits verwendete Framework *Text Mesh Pro* (TMP) näher betrachtet. Letztlich wurde sich für dieses Framework entschieden, da es bereits für das bestehende Code-Window verwendet wird.

TMP: Ein Framework zur Ein- und Ausgabe von Texten in Unity.

Dadurch, dass TMP schon verwendet wird, ist es dann einfacher, das Code-Window anzupassen. Noch wichtiger ist jedoch, dass das bestehende Syntax-Highlighting über *Rich-Text*[3] dargestellt wird.
Dieser ist aktuell an TMP angepasst und kann bei einer Weiterverwendung des Frameworks ohne große Anpassungen übernommen werden. Des Weiteren unterstützt TMP auch mehrzeilige Texteingaben, was wichtig ist, da die Dateien in der Regel aus mehr als einer Zeile bestehen.

Rich-Text: Besteht aus XML-Tags. Diese XML-Tags enthalten dann beispielsweise Farbcodes wie $< color = \#005500 > DarkGreen < /color >$. Der Text *Dark Green* wird damit in dunkel Grün dargestellt. [2]

[3] Eine komplette Einführung in die möglichen Formate mit Rich-Text in TMP gibt es hier: `http://digitalnativestudios.com/textmeshpro/docs/rich-text/` Hinweis: Das XML Beispiel stammt aus der zuvor genannten Quelle

4.2 Grundlegende Editor Funktionen

Ein gängiger Texteditor sollte mindestens die folgenden zwei Funktionen ermöglichen, damit mit diesem sinnvoll gearbeitet werden kann: Zum einen muss eine Lösung zum Speichern des Inhaltes gefunden werden und zum anderen muss ein Benutzer Änderungen rückgängig machen können.
Beide Funktionen werden im Folgenden genauer beschrieben und konzeptioniert.

4.2.1 Speichern von Änderungen

Eines der wohl wichtigsten Funktionen eines Texteditors ist das Speichern der Eingaben vor Beendigung der Anwendung. Ein Editor, der Inhalte nur zur Laufzeit speichert, würde lediglich für kurzzeitige Notizen Sinn ergeben, was in diesem Fall allerdings nicht genügen würde.
Dementsprechend muss ein Konzept zur Speicherung des Inhaltes eines Code-Windows entwickelt werden. Dies ist einfach umzusetzen, da der Inhalt des Code-Windows mit wenig Aufwand vom Rich-Text getrennt werden kann und der Text dann mithilfe einer C# Funktion in die entsprechende Datei geschrieben werden kann. Dabei muss zunächst die Zieldatei bekannt sein, in die gespeichert werden soll. Als Zieldatei wurde standardmäßig die Datei definiert, die auch als Quelldatei dient. Dort wird der Inhalt nach Speicherung überschrieben. Eine Funktion zur Auswahl des Speicherortes ist zunächst nicht geplant, da es zusätzlichen Aufwand bedeutet und die Zeit zunächst für die Kernfunktionalitäten verwendet werden soll.

4.2.2 Rückgängig machen von Änderungen

Da die meisten modernen Systeme es dem Benutzer erlauben, geschehene Aktionen rückgängig zu machen, soll dieser Editor auch dies unterstützen. Das Rückgängig machen einer Aktion wird auch als Undo und die erneute Ausführung rückgängig gemachter Aktionen als Redo bezeichnet. Diese zwei Operationen sind im Editor-Kontext besonders wichtig. Ein Beispiel für die Relevanz dieser Funktionalität kann das versehentliche Löschen eines Wortes darstellen. Sollte der Nutzer diese Löschung rückgängig machen wollen, könnte er über ein Undo das gelöschte Wort wieder hinzufügen. Ein Redo würde dann die erneute Löschung bewirken. Das Hinzufügen von Texten lässt sich zwar theoretisch einfach durch das Drücken der Löschtaste rückgängig machen, wird gleichwohl meist auch durch das Undo-Konzept abgedeckt.

Für das Undo/ Redo gibt es bereits etablierte Konzepte. In dieser Implementierung soll das Konzept eines Memento verwendet werden. [Kos19] Für ein Memento wird der Zustand eines Objektes zwischengespeichert, sodass er später

erneut abrufbar ist. Dabei ist der Zustand des Objektes für andere nicht einsehbar. Dieses Objekt wird auf einem *Stack* gespeichert. Dieser funktioniert analog zu einem Stapel in der realen Welt, es kann immer nur auf das oberste Objekt des Stapels zugegriffen respektive das oberste Objekt gelöscht werden. Auch können neu erzeugte Objekte immer nur oben auf dem Stapel hinzugefügt werden. C# hat standardmäßig einen *Stack* im Funktionsumfang, der für das Undo und Redo benutzt werden kann.[4]
Möchte jetzt also ein Nutzer den alten Zustand wiederherstellen, wird das erste Objekt vom *Stack* genommen und als aktuelles geladen. Um danach noch ein Redo auszuführen, wird der Zustand des Objektes vor dem Undo dann auf einem zweiten *Stack* gespeichert, dem *Redo-Stack*. Der *Redo-Stack* wird allerdings immer dann komplett geleert, wenn der Nutzer eine neue Aktion ausführt, da diese sonst mit dem Redo in Konflikt geraten könnte. So können die Zustände dann beliebig vor- und zurückgestellt werden.
Dieses Konzept wird allerdings komplizierter, wenn mehrere Benutzer den Zustand der Objekte ändern können, der Editor also im Mehrbenutzer-Betrieb parallel genutzt wird. Dann könnte es nämlich passieren, dass zum Beispiel zuerst ein Nutzer A den Namen eines Objektes von Bar auf Foo geändert hat, allerdings ein weiterer Nutzer B dann den Titel auf $FooBar$ ändert. Wenn nun A seine Änderung rückgängig machen möchte, wird sie von der Änderung durch B blockiert. Dieses Problem zu lösen ist etwas komplexer. Bei Objekten mit mehreren Eigenschaften müsste dann entweder entschieden werden, ob die Änderung von B dieselbe Eigenschaft betrifft, wie die von A und es dem entsprechend angepasst werden, oder das Undo des Nutzers A muss blockiert werden, solange die Änderung von Nutzer B in der Historie existiert. Eine Lösung für dieses Problem wurde für SEE bereits an anderer Stelle gelöst (siehe [al21a] (Seite 58-62)). Diese Lösung soll für das Undo/Redo des integrierten Texteditors übernommen werden.

4.3 Syntax Highlighting

Syntax-Highlighting ist aus modernen IDE's nicht mehr wegzudenken, da es die Übersichtlichkeit enorm steigert. Aus diesem Grund soll es auch für den integrierten Texteditor übernommen werden.

Im Allgemeinen kann die Hervorhebung durch Schriftarten, Größen, Stile oder andere Auffälligkeiten geschehen. Im Code-Window werden die Wörter bezüglich ihrer Bedeutung innerhalb einer Programmiersprache oder eines Da-

[4] C# Stack https://docs.microsoft.com/de-de/dotnet/api/system.collections.stack?view=net-6.0

teiformats hervorgehoben.

Das Syntax-Highlighting innerhalb des Code-Windows ist bereits in SEE integriert worden. Hierfür wird das Framework *Antlr*[5] genutzt, um die Bedeutung der Wörter innerhalb einer Datei zu ermitteln. Dazu wird die verwendete Programmiersprache des Quellcodes, und damit seine spezifischen Schlüsselwörter an der Dateiendung erkannt.[al21a] (Seite 25)
Mithilfe von *Antlr* werden *Lexer* generiert, den daraus resultierenden *Token* kann eine Farbe entsprechend ihrer Bedeutung zugewiesen werden und diese mit Rich-Text im Code-Window angezeigt werden. Das TMP Eingabefeld wird dann den eingegebenen Rich-Text in z. B. eine Farbe umwandeln.

Lexer: Ist die Abkürzung für lexikalischer Scanner. Dieser zerteilt einen Text in Token. [al21b]

Token: Ist ein atomarer Teil des zerteilten Quelltextes. [al14]

Dieser Prozess findet aktuell nur beim Laden der Datei statt. Fügt ein Nutzer ein Wort hinzu, wird dies nicht aktualisiert. Es ist allerdings geplant, dass dieser Prozess nach einer Änderung neu ausgeführt werden soll, allerdings eine Pause von ca. 30 Sekunden nach dem Berechnen einlegen soll, damit nicht nach jedem geänderten Buchstaben die komplette Datei neu berechnet werden muss.

4.4 Mehrbenutzerfähigkeit

Im Kern eines mehrbenutzerfähigen Texteditors steht, dass jeder Benutzer den gleichen Quelltext vorliegen haben soll, unabhängig davon, welcher Benutzer wann, was, wo einfügt. Es ist also eine Synchronisierung erforderlich. Deswegen müssen die einzelnen, lokalen Versionen vereint werden. Im Folgenden soll zunächst auf das Problem selbst und dann auf die möglichen Lösungen hierzu eingegangen werden.

4.4.1 Synchronisierung von Texten

Die Synchronisierung von Texten ist keine triviale Aufgabe. Denn die Bearbeitung von Texten durch Einfügen eines Buchstabens an der Position x ist nicht kommutativ — es macht also einen Unterschied, in welcher Reihenfolge Zeichen eingefügt werden. Dies lässt sich mit folgendem Beispiel erläutern:
Die einfüge Operation sei folgendermaßen definiert: $Insert(string,\ idx)$
$String$ steht für das Wort und Idx für die Position, an der das Wort eingefügt werden soll. [Che17] Wenn nun zwei Nutzer gleichzeitig folgenden Satz bearbeiten wollen:

[5] https://www.antlr.org/

Abbildung 5: Beispielsatz eins, zur Veranschaulichung von Synchronisierungsproblemen.

Nutzer A fügt ein „ $süss$“ an das Ende.
Nutzer B fügt ein „$keine$ “ hinter „$ich\ mag$ “.

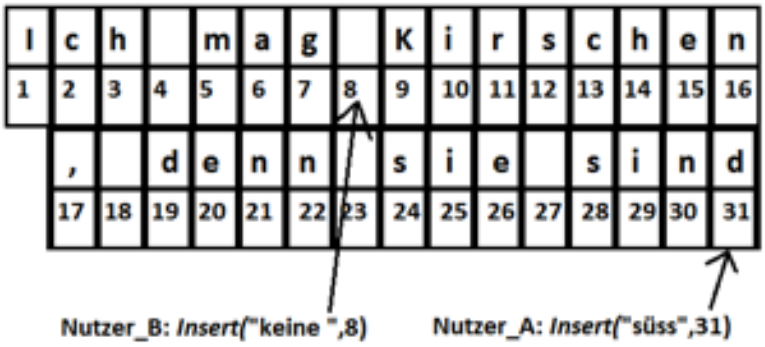

Abbildung 6: Beispielsatz zwei, zur Veranschaulichung von Synchronisierungsproblemen.

Solange A seine Operation zuerst beendet, gibt es kein Problem, wenn aber Nutzer B zuerst „$keine$ “einfügt, gibt es ein Problem. Nun ist es nämlich so, dass die Position, an der Nutzer A „ $süss$“einfügen möchte, nicht mehr 31, sondern 37 ist.

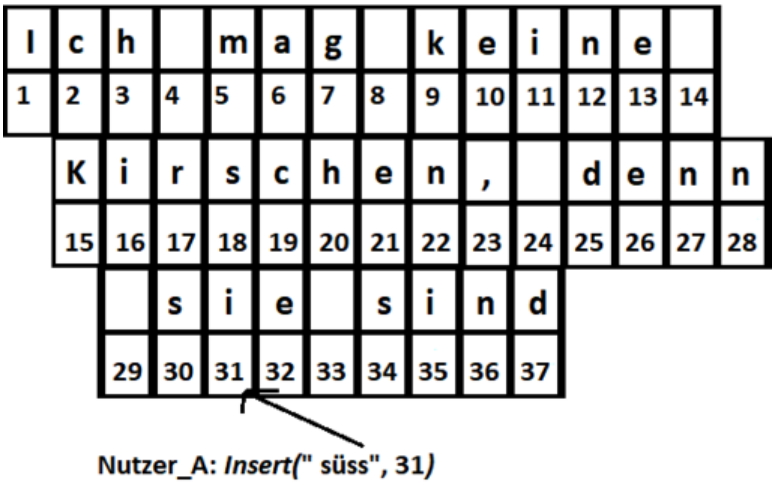

Abbildung 7: Beispielsatz drei, zur Veranschaulichung von Synchronisierungsproblemen.

Daraus würde dann also der folgende Satz entstehen:

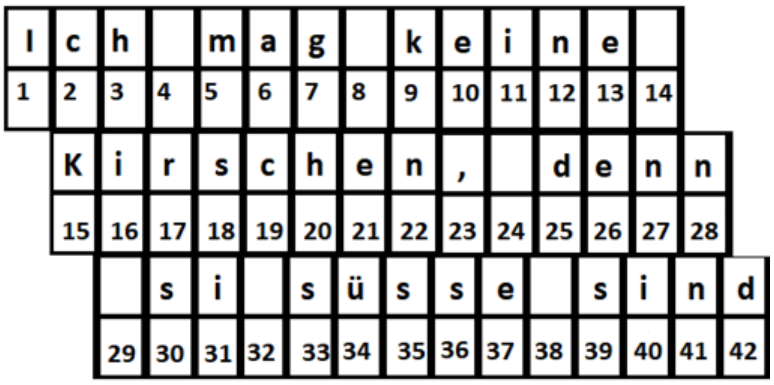

Abbildung 8: Beispielsatz vier, zur Veranschaulichung von Synchronisierungsproblemen.

Der nun entstandene Satz ergibt so natürlich keinen Sinn und ist nicht das gewünschte Ergebnis.

Neben der Kommutativität, muss die Vereinigung zweier Texte auch idempotent sein. Dies gilt für das Löschen von Buchstaben. Idempotent, heißt in diesem Fall, wenn zwei Nutzer denselben Buchstaben löschen, dass dieser nur einmal gelöscht wird und nicht auch noch der darauffolgende.

Auch hier ein kurzes Beispiel:
Die Löschoperation wird wie folgt definiert: $DeleteChar(Idx)$, wobei Idx für die Position steht, die gelöscht werden soll.[Che17]
Nutzer A und B möchten in dem Wort „$Petemr$“ das überflüssige m löschen.
Dazu führen beide die Operation $DeleteChar(5)$ auf.
Nun wird aus „$Petemr$“ „$Pete$“
Fälschlicherweise könnte man nun annehmen, dies lasse sich einfach über das Hinzufügen des Buchstabens in der Operation beheben.
Diese Änderung kann zwar gewisse Probleme lösen, ist aber an dem Punkt begrenzt, wo dasselbe Zeichen mehrfach hintereinander vorkommt, wodurch es zu falscher Übereinstimmung des übergebenen Zeichens kommen kann.

Die Lösung dieser Probleme wurde in der Forschung bereits häufig thematisiert, da sie für alle mehrbenutzerfähigen Texteditoren gültig sind und daraufhin einige Lösungsansätze entwickelt.
Im Folgenden werden nun zwei der bekanntesten Ansätze näher beleuchtet.

4.4.2 Operational Transform

Dieses Problem kollaborativer Textbearbeitung wurde schon früh entdeckt, so entstanden die ersten Lösungsansätze und Modelle bereits 1998.
Ellis & Gibbs entwickelten ein Modell namens dOPT, welches dem Prinzip des Operational Transforms entspricht. [Kle18]
Diese Datenstruktur wird zum Beispiel von *Google Docs* und *Microsoft Office Online* genutzt. Operational Transform ist allerdings relativ komplex, so entstanden von 1989 an regelmäßig neue Arbeiten, welche die vorherigen zumindest in Teilen widerlegten. [Kle18] Auf Grundlage dieser Widerlegungen entstand bereits im Jahr 1996 die nächste Version — adOPTed, das von Ressel et al. entwickelt wurde. Dieses stellt dar, dass mit dem Algorithmus von 1989 nicht zwangsläufig jeder Benutzer das gleiche Dokument vorliegen hat. Dieses neue Modell wurde dann im Jahr 1997 von Suleiman et al. allerdings erneut widerlegt.

Dieser Prozess setzte sich bis ins Jahr 2006 fort. Von den von Martin Kleppmann [Kle18] aufgeführten acht Arbeiten wurden fünf als falsch bewiesen und zwei von den als noch korrekt geltenden benötigen einen zentralen Server, der die Klienten verwaltet. [Kle18].

4.4.3 Conflict Free Replicated Datatype

Eine deutlich weniger komplexe Variante wurde ab dem Jahr 2006 entwickelt [Kle18]. Der sogenannte *Conflict Free Replicated Datatype* (CRDT) wurde etwa von *Teletype for Atom* und *Redis* benutzt.

CRDT: Ein Ansatz, um zwei oder mehr Versionen eines Textes zusammen zuführen.

Es gibt verschiedene Implementierungen, zumeist in JavaScript verfasst.
Im Rahmen der Arbeit von Martin Kleppmann [Kle18] wurde beispielsweise *Automerge* entwickelt — die Intention hinter dieser Arbeit war die Erstellung eines formalen Beweises des CRDT's. *Automerge* ist eine Plattform auf der verschiedene mehrbenutzerfähige Anwendungen gebaut werden können. So haben die Entwickler selbst zum Beispiel *Trellis*, eine Art Kanban Board entwickelt oder auch eine App namens *PixelPushers*, in der Pixelart kollaborativ entwickelt werden kann. [Kle18]
Eine weitere Implementierung in Hinsicht auf einen kollaborativen Texteditor hat Rudi Chen [Che17] entwickelt. Basierend auf dessen JavaScript-Auszügen soll das CRDT in C# für SEE entwickelt werden.

Die Idee dahinter ist, dass jeder Buchstabe eine ID erhält. Über diese ID kann seine Position eindeutig bestimmt werden, da die IDs in aufsteigender Reihenfolge vergeben werden.
Als Intuition könnte man zunächst einmal annehmen, dass die Indizes als ID verwendet werden. Eine Löschoperation würde nun nur diese ID bekommen. Da die ID nach dem Löschen nicht verschoben wird, wird nur ein Buchstabe gelöscht, auch wenn mehrere Benutzer gleichzeitig versuchen, ihn zu löschen. Beim Einfügen eines neuen Zeichens, welches nicht am Ende eingefügt wird, bestehen allerdings neue Probleme. Ein Beispiel hierfür wäre das Verhalten, wenn ein Buchstabe zwischen der ID 2 und 3 eingefügt werden soll.
Ein intuitiver Ansatz zur Lösung wäre die Nutzung von Gleitkomma-Zahlen, wie beispielsweise $2,5$. In der Implementierung werden allerdings statt dieser Gleitkomma-Zahlen ID-Listen verwendet. Im konkreten Beispiel würde der Buchstabe zwischen 2 und 3 die ID $\{2,1\}$ erhalten, es wird also ein zweiter Index eingefügt, welcher ab der eingefügten Position beginnend gezählt wird. Diese ID-Liste wird bei jeder neuen Änderung also ein neuer Index angehängt.

4.4.4 Vergleich der Ansätze

Nach der Begutachtung der beiden Ansätze, wurde sich für die Lösung mittels CRDT entschieden, da diese Variante deutlich simpler in der Implementierung und infolgedessen besser und zuverlässiger zu integrieren ist.

4.4.5 Schnittstelle zwischen den Klienten

Die Idee hinter diesem Ansatz ist, dass das CRDT dafür sorgt, dass die Texte aller Benutzer zu jedem Zeitpunkt gleich sind. Deshalb muss es eine Schnittstelle der lokalen CRDTs geben, sodass diese ihre Inhalte austauschen können. Um diesen Prozess möglichst effizient zu gestalten, soll jede Änderung an dem lokalen CRDTs direkt zu den CRDTss der anderen Benutzer übertragen werden.

Während Änderungen an dem lokalen Eingabefeld mit dem Index als Position in das CRDT eingefügt werden können, muss zur Synchronisation mit den anderen zunächst die oben erklärte ID berechnet werden. Diese ID kann dann zusammen mit dem Buchstaben als Änderung zu den anderen Klienten geschickt werden. Diese müssen dann nur noch die erhaltene ID mit den IDs, die bereits im CRDT stehen, vergleichen und den Buchstaben zusammen mit der ID dann unmittelbar vor der nächst größeren einfügen.

Um diese Änderungen zu verschicken, kann die bereits existierende Netzwerkverbindung zwischen den Klienten erweitert und genutzt werden. Dabei muss lediglich sichergestellt werden, dass das CRDT bei den anderen Klienten lokalisiert werden kann. Dafür soll ein statisches Interface genutzt werden, in dem die entsprechenden CRDTs über den Dateinamen, zu dem sie gehören, wiedergefunden werden können oder, falls keines existiert, lokal erstellt werden können.

4.5 Schnittstelle zur GUI

Nachdem nun die zwei Komponenten GUI und CRDT entworfen wurden, musste noch eine Schnittstelle zwischen den beiden geschaffen werden. Dazu wird ein statisches Interface verwendet, welches zu jedem geöffneten Code-Window ein CRDT erstellt. In der Schnittstelle werden allerdings nur die Änderungen im Text als Eingabe erwartet, das heißt ein $AddString$ oder $DeleteString$. Um diese Informationen aus dem Texteingabefeld zu erhalten, könnte bei einer Änderung der gesamte Text aus dem Eingabefeld genommen werden, und mit dem alten Text, dem vor den Änderungen, verglichen werden. Diese Lösung ist aber aus verschiedenen Gründen nicht optimal. Zum einen ist es bei einer simplen, Zeichen für Zeichen vergleichenden, Version nur möglich, eine einzige Änderung zu erkennen, wie ein Löschen oder Hinzufügen, nicht aber multiple Änderungen oder auch ein Ersetzen von Zeichen. Zum anderen muss für jede Änderung immer der gesamte Text bis zur Position der Operation geparst werden. Da Dateien häufig viele Zeichen enthalten und diese bei jeder Änderung durchgegangen werden müssten, könnte dies in Laufzeit- und Leistungsproblemen resultieren. Deswegen muss eine Alternative gefunden werden.

4.5.1 Input Listener

Eine mögliche Alternative bietet die Verwendung von sogenannten $Input\ Listener$. Dies bezeichnet ein Stück Code, welches immer wieder aufgerufen wird und nur darauf wartet, dass eine Eingabe erfolgt. Sobald eine Eingabe stattfindet,

gibt der
$Input\ Listener$ die Information über diese Eingabe weiter. In diesem Fall müssen also die gedrückten Buchstaben abgefangen werden und an das CRDT weitergegeben werden. Da das CRDT allerdings auch die Position benötigt, an der eine Änderung stattgefunden hat, muss diese zusätzlich erkannt werden. Dies kann aber über die Cursor-Position im Eingabefeld ermittelt werden. Dieses Konzept funktioniert allerdings nur, wenn Buchstaben hinzugefügt werden. Sollten Buchstaben gelöscht werden, kann die gedrückte Taste $Backspace$ oder $Delete$ nicht in das CRDT eingefügt werden. Stattdessen werden diese Tasten gesondert erkannt und ein dementsprechender $DeleteString$ Befehl ausgelöst. Hierbei muss dann unterschieden werden, ob $Backspace$ oder $Delete$ gedrückt wurde, da ersteres den Buchstaben links des Cursors löscht und letzteres das Zeichen rechts des Cursors.
Falls Tastenkombinationen wie $CTRL + S$ gedrückt werden, müssen auch diese abgefangen werden, da in diesem Fall kein s eingefügt, sondern der Inhalt abgespeichert werden soll.

4.5.2 Aktualisierung der GUI

Nachdem nun feststeht, wie Änderung von der GUI in das CRDT kommen, muss abschließend noch geklärt werden, wie diese Änderungen bei anderen Klienten vom CRDT aus in die GUI gelangen können.
Intuitiv wäre der Ansatz, dass der Inhalt des CRDT in eine Zeichenkette geschrieben wird und diese dann wiederum in das Eingabefeld. Dies ist aber auch aus Gründen der Laufzeit nicht möglich, da das Eingabefeld nicht für eine solche Menge an Datenänderungen geeignet ist.
Es kann davon ausgegangen werden, dass, wenn der $Input\ Listener$ korrekt umgesetzt wurde, im CRDT exakt die gleichen Daten stehen, wie auch im Eingabefeld. Dementsprechend muss lediglich eine Änderung, welche durch einen anderen Teilnehmer im Eingabefeld vorgenommen wurde, in das Eingabefeld des lokalen Benutzers eingefügt werden. Da das CRDT diese fremden Änderungen bereits einzeln empfängt, müssen diese lediglich an das Code-Window weitergereicht werden.
Da im CRDT selbst allerdings keine Referenz zum Code-Window existiert, wird dazu der $CodeSpaceManager$ der bereits bestehenden Grundlage verwendet.
Der $CodeSpaceManager$ ist bisher dafür zuständig gewesen, die verschiedenen Code-Windows zu verwalten. Dabei hat er unter anderem dafür gesorgt, dass, wenn ein Klient die Ansicht des Code-Windows eines anderen Klienten gewählt hat, dessen Code-Window durch das andere ferngesteuert werden kann, um die bereits angesprochene Synchronisierung des Ausschnitts zu bewirken. Darüber sollte ein Eingriff in den Inhalt des Code-Windows möglich

sein.

4.6 Fazit der Konzeption

Zusammenfassend wird also ein Eingabefeld von TMP benutzt, um eine Texteingabe zu ermöglichen. Des Weiteren soll der Editor Änderungen speichern können und ein Undo/ Redo unterstützen, für beides sollen bereits existierende Klassen benutzt werden. Das aktuell bestehende Syntax-Highlighting soll dahin gehend erweitert werden, dass es nach Änderungen neu berechnet wird. Zuletzt soll die Mehrbenutzerfähigkeit durch die Verwendung eines noch zu implementierenden CRDT's und die Synchronisierung dieses mit der GUI durch die Verwendung von *Input Listener* integriert werden.

Kapitel 5 IMPLEMENTIERUNG

Im folgenden Kapitel wird die Integration der in Abschnitt 4 vorgestellten Konzepte in SEE beschrieben.
Zunächst wird erklärt, wie die grafische Benutzeroberfläche angepasst wurde, im Anschluss wird dann die Implementierung der grundlegenden Editor-Funktionen erläutert. Darauf folgte die Einrichtung der Mehrbenutzerfähigkeit, genauer die Synchronisation des Inhalts zwischen den einzelnen Nutzern. Abschließen wird dann auf Ladezeit Probleme und die Testung der Implementierung eingegangen.
Der Quellcode von SEE und die in diesem Abschnitt beschriebene Implementierung kann unter `https://github.com/uni-bremen-agst/SEE` gefunden werden. Die hier beschriebenen Änderungen sind im Branch Creating-a-Multiuser-Codeinput einzusehen. [6]

5.1 Entwicklung der grafischen Benutzeroberfläche

Zuerst wurde das bestehende Code-Window so umgebaut, dass es das Hinzufügen und Löschen von Buchstaben, sowie ein Editieren des vorhanden Textes ermöglicht. Im Anschluss wird beschrieben, wie mit den vorhandenen Zeilennummern umgegangen und das Syntax-Highlighting erweitert wurde.

5.1.1 Ermöglichen von Textbearbeitung

Zunächst musste eine Einarbeitung in die bestehende Codebasis erfolgen. Dazu wurde, der Autor und Entwickler der bestehenden Code-Windows um eine Schulung gebeten. Nach der Durchführung dieser wurde daraufhin ein Eingabefeld in das *Prefab* des Code-Windows integriert. Dieses funktionierte auch zunächst — einfacher Text konnte eingegeben werden. Allerdings war mit der Standard-Einstellung des Eingabefelds nur eine einzeilige Texteingabe möglich. Dies konnte allerdings mithilfe einer Konfiguration des Prefabs umgestellt werden. Allerdings resultierte dies in neuen Problemen.
Es musste nun ein *Viewport* für das Eingabefeld festgelegt werden. Dazu gab es einige Möglichkeiten. Nachdem alle vorgeschlagenen Viewports durchprobiert wurden, konnte das Problem allerdings nicht vollständig gelöst werden. Konkreter funktionierte das Scrollen durch das Eingabefeld nicht richtig. Das Mausrad funktioniert nur zum hoch Scrollen und wenn der Scrollbalken verwendet wird, wird wie in Abbildung 9 die Selektion verschoben und der Cur-

Prefab: Ein Unity-Objekt, welches GUI-Elemente enthält und mithilfe von C# Skripten beliebig oft in Szenen eingefügt werden kann.

Viewport: Der Bereich, in dem der Inhalt angezeigt wird.

[6] Dieses Repository ist nicht öffentlich. Um Zugriff zu erhalten, nehmen Sie bitte Kontakt mit Prof. Dr. Rainer Koschke (koschke@uni-bremen.de) auf.

sor wie in Abbildung 10 zuerkennen aus dem Code-Window hinausgeschoben.

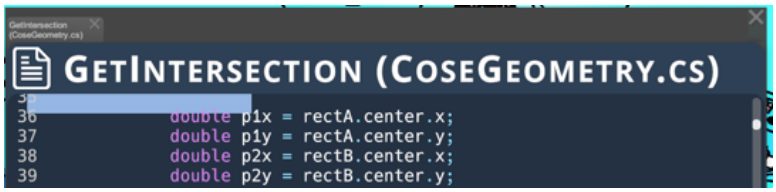

Abbildung 9: Verschobene Selektion nach scrollen im Code-Window.

Abbildung 10: Cursor außerhalb des Code-Windows.

Diese beiden Probleme treten nicht auf, wenn der Nutzer statt des Scrollbalkens die Pfeiltasten benutzt Die Ursache dieses Problems nicht gefunden und behoben werden.

5.1.2 Umgang mit Zeilennummern und Syntax-Highlighting

Gleichwohl gab es noch eine weitere Herausforderung an dieser Stelle. Die bestehenden Zeilennummern wurden nur in das Eingabefeld integriert. Das heißt, sie sind vom Nutzer editierbar. Da dies nicht gewünscht ist, gab es zwei Überlegungen:

Die erste Überlegung war, die Zeilennummern in ein separates Textfeld zu schreiben. Dies wäre allerdings sehr aufwendig und vermutlich nicht im geplanten Zeitrahmen zu bewältigen.
Die zweite Option wäre, die Zeilennummern zu löschen. Dies ist sehr simpel in der Umsetzung, jedoch werden diese Zeilennummern an anderer Stelle der Implementation benötigt und können deswegen nicht gelöscht werden.
Aufgrund dessen wurde sich für keine der beiden Optionen entschieden, da sie nicht in den Zeitrahmen passen.

Dadurch mussten allerdings zwei Dinge beachtet werden. Erstens mussten vor dem Speichern des Inhalts in eine Datei die Zeilennummern entfernt werden, zweitens muss der Text automatisch eingerückt werden, wenn der Nutzer eine neue Zeile anfängt. In der folgenden Abbildung 11 ist diese Einrückung unterhalb der Zeile vier dargestellt.

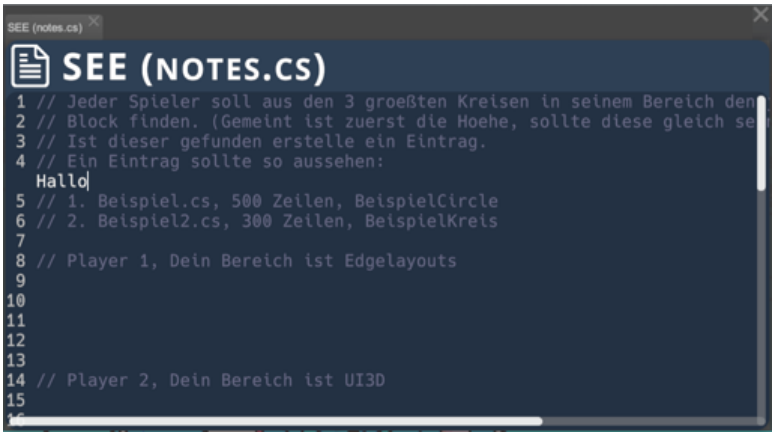

Abbildung 11: Nach dem Drücken von Enter wird der Text automatisch hinter die Zeilennummern eingerückt.

Es wäre zwar gut, wenn vor der Einrückung noch eine aktualisierte Zeilennummer stehen würde, dies ist aber mit dem bestehenden System nicht in dem Umfang umsetzen.
Der Benutzer hat jedoch in der aktuellen Variante die Möglichkeit, die Zeilennummern zu aktualisieren. Dies geschieht zusammen mit dem Syntax-Highlighting. Da die aktuelle Version dieses Syntax-Highlightings den kompletten Inhalt des Code-Windows neu berechnet, wird dies nur auf Verlangen des Nutzers, durch Drücken von $CTRL + R$ durch geführt.

5.2 Grundlegende Editor Funktionen

Im Anschluss an die Überarbeitung der GUI werden nun die grundlegenden Funktionen des Editors implementiert. Hierzu wurden wie in Abschnitt 4 erwähnt zunächst eine Funktion zum Abspeichern der Änderungen und darauffolgend das Undo Redo eingebaut.

5.2.1 Speichern der Änderungen

Zur Speicherung von Änderungen durch den Nutzer wird die häufig verwendete Tastenkombination $STRG + S$ benutzt.
Um die Speicherung umzusetzen, wird lediglich der Text aus dem CRDT extrahiert. Das CRDT bietet hierfür eine Schnittstelle an. Allerdings müssen, an-

ders als in der Konzeption erwähnt, zusätzlich noch die Zeilennummern entfernt werden, denn diese können, wie in Unterunterabschnitt 5.1.2 beschrieben, nicht aus dem Inhalt des Eingabefelds entfernt werden. Nach der Entfernung muss der String dann nur noch in die Datei geschrieben werden, aus welcher der Quellcode zuvor ausgelesen wurde. Dazu konnte eine Standard-Funktion von C# $WriteAllText()$[7] genutzt werden.

5.2.2 Undo und Redo

Wie zuvor in der Konzeption erwähnt, existiert in SEE bereits eine Lösung für das Undo/ Redo Problem. Ursprünglich war es geplant, für den Editor dieses Framework zu benutzen. Dies ist allerdings nicht möglich, da das Framework auf die *SEE-Action*s ausgerichtet ist. Da das Einfügen oder Löschen von Buchstaben im Editor keine klassische SEE-Action ist, wäre es ein großer Aufwand gewesen, dies kompatibel zu machen.

SEE-Action: Sind Aktionen, die ein Nutzer innerhalb von SEE ausführen kann. Hierzu zählt etwa das Löschen eines Gebäudes aus der Software-Stadt.

Ein Grund dafür, dass die aktuelle Version des Undos nicht kompatibel ist, ist, dass dort immer das komplette Game-Objekt als Zustand gespeichert wird. In Falle des Code-Windows wäre es das komplette Fenster. Dies ist allerdings nicht notwendig und es würde sich später blockieren, da im bestehenden Framework eine Undo-Operation komplett blockiert wird, wenn ein anderer Nutzer das gleiche Game-Objekt bearbeitet. Es könnte auch so sein, dass jeder Nutzer sein eigenes Game-Objekt hätte, allerdings würde dann vom Framework nicht erkannt werden, ob die Aktion von Nutzer A die von Nutzer B behindert.
Deswegen wurde eine eigene Lösung entwickelt.
Bei einem Code-Editor treten nämlich viele Probleme, die bei anderen Stellen in SEE auftauchen, nicht auf. Da es nur zwei Aktionen gibt, die rückgängig gemacht werden müssen, ist dieses wesentlich übersichtlicher als das Undo/ Redo für die anderen SEE-Actions. Zudem ist auch die Aktion $DeleteChar$ im Mehrbenutzer-Umfeld unkritisch, da, wenn der Buchstabe von einem Nutzer gelöscht wurde, er von niemanden mehr verändert werden kann. Ein wenig anders sieht es bei $AddChar$ aus, da der Buchstabe von einem anderen Nutzer gelöscht werden könnte.
Dies ist aber kein großes Problem, da in diesem Fall der Undo-Prozess einfach abgebrochen werden kann, da es einfach festzustellen ist, ob der Buchstabe genauer gesagt seine ID schon gelöscht wurde.
Deswegen wurde ein klassisches Undo-/ Redo-Konzept mit zwei *Stacks* umgesetzt. Um eine Aktion rückgängig zu machen, muss nur die komplementäre Aktion ausgeführt werden, also bei $AddChar$ muss $DeleteChar$ ausgeführt

[7] C# *WriteAllText()* `https://docs.microsoft.com/de-de/dotnet/api/system.io.file.writealltext?view=net-6.0`

werden. Die Informationen, die benötigt werden, um nach $DeleteChar$ wieder ein $AddChar$ auszuführen, werden mit auf dem *Stack* gespeichert.

5.3 Synchronisierung von Texten

Zur Umsetzung der Synchronisierung wird das in Unterunterabschnitt 4.4.3 erklärte CRDT verwendet. Zur besseren Verständlichkeit wird hier zunächst die Implementierung als UML-Diagramm abgebildet.

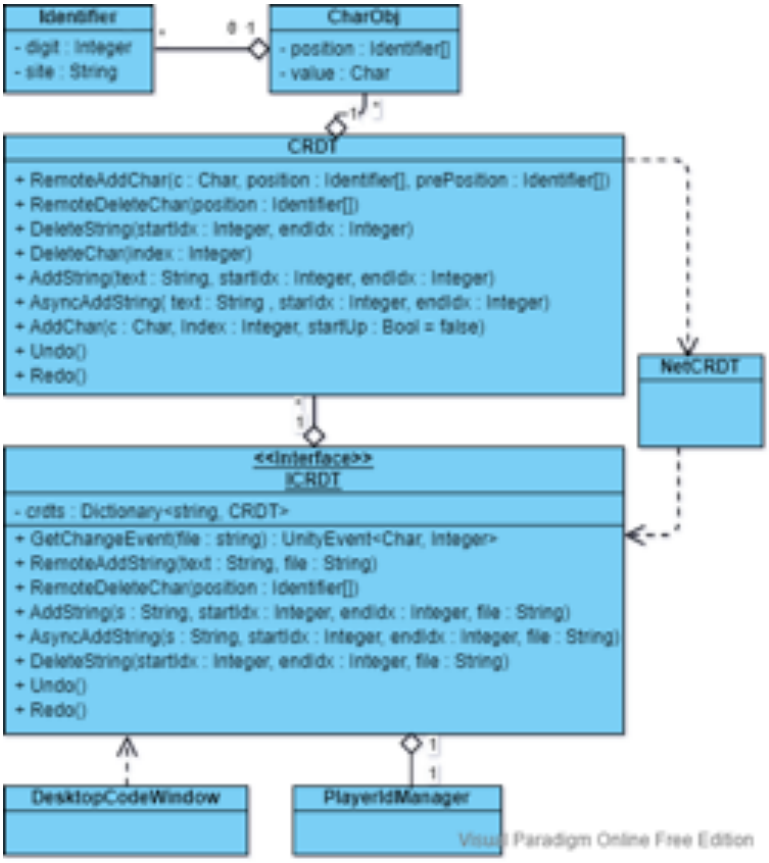

Abbildung 12: Die wichtigsten Komponenten als UML-Diagramm.

Zunächst wird das CRDT mit dessen Komponenten betrachtet.

Grundsätzlich wird im CRDT ein einzelnes Zeichen nicht als Buchstabe betrachtet, sondern als abstraktes Buchstaben-Objekt. Dieses wurde in der Implementierung als CharObj bezeichnet. Dieses Buchstaben-Objekt besteht einerseits aus dem Buchstaben, andererseits aus dem Identifier, der ID des Objektes.

Die ID selbst besteht ebenfalls aus zwei Teilen. Es ist ein Tupel aus einer Position, genauer ein relativer Index, als $digit$ bezeichnet und einer Benutzer-ID.
Die Benutzer-ID, in der Implementierung auch als $site$ bezeichnet, ist pro Benutzer einzigartig und statisch. Sobald ein Nutzer in den *Show-Code-Modus* wechselt, wird eine ID festgelegt und bleibt, bis das Programm beendet wurde, bestehen.
Die Benutzer-ID war zuerst eine GUID[8], da bei dieser sichergestellt ist, dass diese einzigartig ist. Da diese GUID allerdings 16 Byte groß ist und bei jedem Buchstaben mehrfach benutzt wird, wird die GUID stattdessen durch einen Integer ersetzt, der durch den Server verwaltet wird.
Diese Verwaltung erzeugt IDs. Auf dem Server wird ein Integer hochgezählt und beim Klienten dessen ID gespeichert. Dadurch lässt sich die ID sehr kurz und platzsparend halten.

Wie oben erwähnt, repräsentiert der erste Teil des ID-Tupels die Position im Text. Diese entspricht dem Index.
Sobald der Benutzer ein bestehendes Wort editiert, wird der Index für die neuen Buchstaben nicht durch eine Gleitkommazahl, sondern durch einen neuen Index dargestellt. Das Tupel der ID wird dann zu einer Liste aus Tupeln und am Ende wird ein neuer Index angehängt.

Ein Beispiel:
Nutzer A, der die Benutzer-ID 1 hat, schreibt „$Halo$“ in das Eingabefeld. Im CRDT würde jetzt folgendes stehen:
$\{('H', \{(0,1)\}, ('a', \{(1,1)\}, ('l', \{(2,1)\}, ('o', \{(3,1)\}\}$
Wenn Nutzer A nun das fehlende l einsetzt, wird daraus folgendes:
$\{('H', \{(0,1)\}, ('a', \{(1,1)\}, ('l', \{(2,1)\}, ('l', \{(2,1),(1,1)\}, ('o', \{(3,1)\}\}$
ein weiterer Buchstabe hinter dem l könnte dann so aussehen:
$\{('H', \{(0,1)\}, ('a', \{(1,1)\}, ('l', \{(2,1)\}, ('l', \{(2,1),(1,1)\}, ('x', \{(2,1),(2,1)\}, ('o', \{(3,1)\}\}$

Durch dieses Vorgehen gibt es immer eine eindeutige Ordnung innerhalb des CRDTs. Die *Site-ID* stellt sicher, dass diese Ordnung in jedem Fall aufrechterhalten wird. Es kann unter Umständen passieren, dass die *Positions-ID* zweier Einträge gleich ist, wenn zwei Nutzer im selben Moment an derselben Stelle eine Eingabe tätigen.
Dies passiert beispielsweise, wenn ein komplett leeres Dokument von beiden Nutzern gleichzeitig angefangen wird, zu beschreiben. In diesem Fall würden beide Nutzer an Index 0 schreiben. Wenn die Synchronisierung erst danach

[8] C# GUID https://docs.microsoft.com/de-de/dotnet/api/system.guid.newguid?view=net-6.0

passiert, wird bei beiden die *Positions-ID* 0 vergeben. Damit die Buchstaben trotzdem geordnet werden können, wird die Site-ID benutzt, hier ist nämlich eine ID größer als die andere und wird somit weiter hinten einsortiert.

Beim Hinzufügen und Löschen von Buchstaben muss darüber hinaus noch zwischen lokalen und fremden Änderungen unterschieden werden.
Lokale Änderungen können wie bei einem normalen Texteditor gehandhabt werden. Es kann der Index im Eingabefeld zur Positionierung im CRDT genutzt werden.
Auch hier wird eine Position berechnet, diese wird allerdings zunächst nicht benötigt.
Wenn dagegen eine Fremd-Änderung durchgeführt wird, muss eine eindeutige Position bekannt sein.
Deswegen wird der *Identifier* des Buchstaben-Objektes des fremden Klienten als Position übergeben. Dieser *Identifier* kann mit den vorhandenen Einträgen im CRDT verglichen werden und somit der Index, an dem die Änderung eingefügt werden soll, bestimmt werden.

Zu den zwei Fallunterscheidungen bei Änderungen gibt es noch einen Sonderfall. Wenn ein Code-Window das erste Mal geöffnet wird, muss der komplette Datei-Inhalt in das CRDT geschrieben werden. Da dies bei großen Dateien einige Zeit erfordert, wurde eine asynchrone Variante der Hinzufügung der Zeichen erstellt. Diese unterscheidet sich kaum von der synchronen Variante, außer darin, dass auf ein Framework von Unity zurückgegriffen wurde, um diese Funktion im Hintergrund aufrufen zu können.
In der Praxis hat sich diese asynchrone Funktionalität allerdings nur für das Neu-Laden einer Datei bewährt und wird aus diesem Grund nur hier verwendet.

5.4 Schnittelle zwischen GUI und CRDT

In diesem Abschnitt wird erklärt, wie das in Abschnitt 4 entworfene Konzept zur Umsetzung der Schnittstelle zwischen GUI und CRDT implementiert wurde. Im Anschluss wird erklärt, wie die umgekehrte Schnittstelle zwischen CRDT und GUI implementiert wurde.

Event: Eine Unity Klasse, welche unter anderem Peripherie Eingaben erkennt und für andere Programmteile nutzbar machen kann. Zusätzlich können mittels eines Events auch eigene Änderungen zwischen verschiedenen Klassen ausgetauscht werden.

5.4.1 Aktualisieren des CRDT

Zunächst mussten die Eingaben aus der GUI in das CRDT eingefügt werden. Dazu werden sogenannte *Input Listner* verwendet.

Einfügen von Zeichen Ein *Input Listener* kann durch die Unity-Klasse *Event* erstellt und genutzt werden. Diese kann innerhalb einer von MonoBehaviour

erbenden Klasse in der Methode $OnGUI()$ abgefragt werden. Die Event Klasse liefert ein neues Event immer dann, wenn eine Eingabe durchgeführt wird. Auf Grundlage dieses Events kann nun überprüft werden, ob dieses von der Tastatur ausgeführt wurde.
Das Problem an dieser Methode ist allerdings, dass es standardmäßig ein US-Tastaturlayout zurückgibt. Das heißt, dass das Tastaturlayout des Benutzers müsste abgefragt und dann die Tastenbelegung angepasst werden. Wird dies nicht gemacht, werden sonst beispielsweise Buchstaben wie *Ä/Ö/Ü* nicht verfügbar sein und einige Symbole auf der Tastatur an anderen Positionen liegen als aufgedruckt.
Zusätzlich wird bei einem Druck von z.B. 1 auch nicht einfach 1 ausgegeben sondern *Alpha1*. Das bedeutet, dass einige eingaben zusätzlich übersetzt werden müssen. Außerdem wird nicht ausgegeben, ob *Caps* oder *CapsLock* gedrückt wurde, sodass dies manuell angepasst werden müsste.

Da dies ein sehr großer Aufwand ist, wurde eine zweite Recherche durchgeführt. In einem Artikel wurde eine Methode ermittelt, die innerhalb der Unity Input-Klasse implementiert wurde. [Mar19].
Input verfügt über eine Methode $inputString()$, welche die betätigte alphanumerische Taste im richtigen Tastaturlayout zurückliefert.
Damit musste nur noch dafür gesorgt werden, dass der Buchstabe an der richtigen Stelle eingefügt wird.
Dazu wird die Cursorposition im Eingabefeld verwendet. Diese kann mithilfe einer Methode vom TMP ausgelesen werden und ist die Position, an der der Buchstabe eingefügt oder gelöscht werden soll.

Löschen von Zeichen Das Löschen ist wiederum komplexer. Da die oben genannte *Input* Methode keine Ausgabe für eine Löschoperation hat, mussten diese Tasten zusätzlich abgefragt werden. Dazu gibt es zwei triviale Fälle: *Delete* und *Backspace*.
Da diese Tastenabfrage allerdings jeden Frame ausgeführt wird und ein Frame bei 60 FPS ca. 16 ms lang ist, musste noch ein Cooldown eingefügt werden, damit beim Druck einer Taste nicht mehr als ein Zeichen gelöscht wird.
In späteren, ausführlichen Tests ist dann aufgefallen, dass ein Cooldown nicht alle auftretenden Probleme löst.
Wenn die *Backspace* Taste gedrückt gehalten wird, kann es sein, dass das Eingabefeld langsamer ist als die Abfrage im Backend. Somit wurden zu viele Buchstaben im Backend gelöscht.
Daraufhin wurde versucht, den Cooldown zu optimieren. Es war allerdings

nach einigen Versuchen nicht möglich, diesen korrekt einzustellen, da die Zeit innerhalb des Eingabefeldes variiert, in der ein Buchstabe gelöscht wurde. So ist es am Anfang etwa tendenziell langsamer, am Ende tendenziell schneller.

Ein weiteres Problem war trat auf, wenn das Löschen schneller war als die Bewegung des Cursors, somit die Buchstaben rechts vom Cursor gelöscht wurden, wie es bei der *Delete-Taste* üblich wäre.
Nach einigen Theorien und Recherchen ist dann eine Variante entwickelt worden, die ein Signal vom Eingabefeld bekommt, wenn es eine Änderung gab. So konnte das Problem der zu viel gelöschten Buchstaben gelöst werden.
Zusätzlich wurde noch die Cursorposition zwischengespeichert. Wenn nun innerhalb des Cooldowns von wenigen Millisekunden die Taste erneut gedrückt wurde, respektive gedrückt gehalten wurde, wird einfach die Cursorposition mit der alten verglichen und gegebenenfalls entsprechend nach links oder rechts verschoben.
Zuletzt kann es passieren, dass ein Buchstabe im Eingabefeld gelöscht wird, aber nicht im CRDT. Es wird vermutet, dass es an einem ungünstigen Timing liegt, also wenn das Löschen genau bei einem Framewechsel passiert. Hier gibt der Editor die Änderung erst bekannt, wenn der Frame vorbei ist, obwohl die *Backspace-Taste* bereits im vorherigen betätigt wurde.
Als Lösung hierfür wird der letzte Tastendruck zwischengespeichert und in einem solchen Fall erneut ausgeführt. Dieser Fall kann einfach erkannt werden, denn das Eingabefeld gibt ein Änderungssignal, welches allerdings von keinem *Input Listener* genutzt wird.

Abschließend zeigten sich noch zwei weitere Schwierigkeiten.
Es funktionierte zwar die *Delete-Taste* auch bei der Übertragung im Netzwerk, jedoch die *Backspace-Taste* nicht richtig. Durch das Abfangen der Übertragung festgestellt werden, dass bei einem *Backspace* auch der *ASCII*-Code vom $inputString()$ weitergegeben wird. Deswegen muss der eingegebene String dann auf „$\backslash b$“ überprüft und dieses dann entfernt werden.

ASCII: American Standard Code for Information Interchange. Dies ist eine Tabelle, in der festgehalten wird, wie ein Zeichen in verschiedenen Formaten übertragen wird.

Zu den oben genannten trivialen Fällen gibt es noch zwei Sonderfälle, dies es zu lösen gilt. Einerseits kann mit der Maus ein Text im Eingabefeld markiert werden, hier wird dann vom Benutzer erwartet, dass dieser markierte Text bei einem weiteren Tastendruck gelöscht wird. Dazu konnte eine Funktion des Eingabefeldes benutzt werden. Dieses gibt die Index-Grenzen des markierten Textes aus, sodass dieser dann gelöscht werden kann.
Anderseits musste auch ein Einfügen mit $CTRL+V$ möglich sein. Dazu musste die Systemzwischenablage benutzt werden, was mit einer Unity-Methode

realisierbar ist.

Einfügen von Zeilenumbrüchen Zuletzt fiel auf, dass zwar beim Drücken von *Enter* im CRDT ein Absatz erzeugt wird, jedoch im Eingabefeld eines fremden Klienten nur zum Anfang der Zeile gewechselt wird und eine zweite Schicht Text über dem vorherigen Text eingefügt wurde. (Siehe Abbildung 13) Der Grund für dieses Problem war, dass beim Drücken der *Enter-Taste* im Eingabefeld auf Windows-Betriebssystemen ein *carretreturn* $(\backslash r)$ an das Ende der Zeile angefügt wird. Dieses Zeichen wurde daraufhin durch ein $\backslash n$ ersetzt, welches den Zeilenumbruch auslöst.

Wie in Unterunterabschnitt 5.1.2 schon erwähnt, muss beim Beschreiben einer neuen Zeile darauf geachtet werden, dass der Platz, der durch die Zeilennummern belegt wird, nicht mit eigenen Inhalten beschrieben wird.
Deswegen wurde dieser Platz beim Drücken der *Enter-Taste* automatisch mit Leerzeichen aufgefüllt und der Cursor entsprechend verschoben.

Abbildung 13: Fehlerhafter Zeilenumbruch.

5.4.2 Aktualisierung der GUI

Wie bereits in der Konzeption erwähnt, sollten nur die Änderungen, die von anderen Klienten stammen, vom CRDT in das Code-Window geschrieben werden. Zur Umsetzung wurden Unity-Events genutzt, diese Events funktionieren ähnlich wie die *Input Listener*, sie warten darauf, dass an anderer Stelle ein Event ausgelöst wird, welches daraufhin verarbeitet werden kann.
In der Implementierung wurde daraufhin ein Event im CRDT erstellt, welches angibt, ob ein Buchstabe hinzugefügt oder gelöscht werden soll und an welchem Index dieser Buchstabe steht. Hinzufügen wird der Buchstabe zusätzlich übertragen, Löschvorgang allerdings nicht, hier genügt die Position des zu löschenden Buchstabens.

Im Code-Window wird dann auf ein solches Event gewartet und der Buchstabe an dem gegebenen Index eingefügt oder gelöscht.

5.5 Ladezeit-Probleme

Schon während der Implementierung der Basisfunktionen fiel auf, dass die Ladezeit von größeren Dateien teilweise mehrere Minuten dauerte. Überdies hängte sich dabei die gesamte Anwendung auf. Da dies die Anwendung nahezu unbenutzbar machte, musste eine Lösung hierfür entwickelt werden.

Zunächst musste ermittelt werden, welche Komponenten diese Ladezeit zu verantworten haben. Bei der Durchsicht des Quellcodes fiel auf, dass aktuell noch eine lineare Suche, zum Finden der geeigneten Position von Fremd-Änderungen, verwendet wurde. Da die lineare Suche eine Laufzeit von $O(n)$ hat, wurde diese durch eine binäre Suche ersetzt. Dieser Einsatz war möglich, da es sich beim CRDT um ein sortiertes Objekt handelt. Dies verbesserte die Laufzeit auf $O(log(n))$. Gerade bei größeren Dateien wirkten sich diese Änderungen positiv aus — Dateien können mitunter bis zu 20´000 Zeichen enthalten.
Auf die initiale Ladezeit wirkten sich diese Änderungen allerdings nicht aus. Eine mögliche Begründung hierfür könnte sein, dass die Suche nur bei Fremd-Änderungen gebraucht wird und beim Öffnen noch keine Fremd-Änderungen vorgenommen werden müssen.
Deswegen wurden dann Unity Debug-Ausgaben mit der Laufzeit an verschiedenen Stellen eingebaut. Damit konnte festgestellt werden, dass der Großteil des Codes relativ effizient funktioniert, also weniger als $1\ ms$ dauerte und mit den Zeitstempeln nicht genauer gemessen werden konnte.
Mithilfe der Debug-Ausgaben konnte allerdings auch festgestellt werden, dass es Probleme bei der Übertragung über das Netzwerk gab.
Bei dem Versand über das Netzwerk wurden die Informationen zunächst in eine Zeichenkette umgewandelt. Dies musste gemacht werden, da die aktuelle Netzwerkarchitektur nur sehr begrenzte Möglichkeiten zur Datenübertragung bietet. Dieser Vorgang nimmt einige Zeit in Anspruch, da bei dem ersten Öffnen einer Datei der gesamte Inhalt in das CRDT geschrieben werden muss und dieser über das Netzwerk an die anderen Klienten geschickt wird.
Ein Lösungsansatz sollte sein, den so versendeten Inhalt zu optimieren und die Ladezeit so zu optimieren. Bei diesem Inhalt fiel zunächst die *SiteID* auf. Die *SiteID* wurde deshalb, wie in Unterabschnitt 5.3 beschrieben, aufgrund der Länge einer GUID, diese durch einen Integer ersetzt. Die Ladezeit verbesserte sich durch diesen Austausch erheblich.
Eine weitere Strategie zur Verbesserung der Leistung stellte die Entfernung des Rich-Textes für das Syntax-Highlighting vor dem Versand über das Netz-

werk dar. Der Rich-Text kann daraufhin bei jedem Klienten lokal neu berechnet werden. Zudem wird der Rich-Text lokal im Code-Window gespeichert, sodass dies eine doppelte Belastung darstellt.
Deswegen wurde eine Änderung vorgenommen, dass nur noch der reine Text im CRDT gespeichert wird. Dies hat die Ladezeit der größeren Dateien auf ca. 20 Sekunden reduziert.

Da dabei allerdings immer noch die GUI eingefroren war, musste dieser Prozess ausgelagert werden, damit die normale Anwendung noch parallel genutzt werden kann.
Dazu wurde das Unity Async Framework verwendet. Dies hatte jedoch den Nachteil, dass die Netzwerkaufrufe nun zunächst in einen Puffer geschrieben wurden und nicht mehr pro Buchstabe geschahen. Erst nach Beendigung des asynchronen Teiles konnte dieser versendet werden. Dadurch hat sich die Ladezeit auf ca. eine Minute bei großen Dateien erhöht. Durch das asynchrone Laden konnte die Ladezeit allerdings weniger wahrnehmbar gemacht werden, da der Code jetzt früher zum Anschauen bereitstand.

5.6 Testen der Implementierung

Zur Überprüfung der Implementierung mussten Tests durchgeführt werden. Die Implementierung wurde auf zwei Arten getestet.

Das CRDT selbst wurde mithilfe von Unit-Tests getestet, dabei wurden alle wichtigen Funktionen mit Tests versehen, die möglichst alle Fälle, die eintreten könnten, abdecken.

Die GUI sowie die Schnittstelle zum CRDT wurde manuell getestet. Da es relativ schwierig und aufwendig ist, die GUI automatisiert zu testen, wurde an dieser Stelle auf automatisierte Testung verzichtet. Die manuellen Tests wurden sowohl allein als auch mit mehreren Personen über das Netzwerk getestet. Die Anwesenheit mehrerer Personen bei den Tests war unter anderem dazu wichtig, realistische Darstellung der Eingaben von mehreren Benutzern simulieren zu können.

5.7 Fazit der Implementierung

Abschließend lässt sich feststellen, dass die Implementierung deutlich komplexer und langwieriger war als geplant. Besonders viele kleine Laufzeit und Probleme in der Konzeption verlängerten diesen Zeitraum erheblich. Ausführliches Debugging der Fehler führte ebenfalls zu langwierigen Testzeiträumen. Besonders durch den Umfang des Codes stellte sich dies häufig als schwierig heraus. Auch der Kompilierprozess und Startprozess der Unity-Anwendung

nach jeder Änderung speziell im Netzwerk mit mehreren Clients erforderte hierbei erheblichen Aufwand.

Letztlich konnten jedoch alle zuvor definierten Funktionen implementiert werden, woraus als Resultat ein mehrbenutzerfähiger Quelltexteditor entstand, der die wichtigsten Grundfunktionen eines Editors erfüllt. Überdies konnte auch die Synchronisierung mehrerer Clients über ein Netzwerk angemessen realisiert werden.

Kapitel 6 EVALUATION

Im Anschluss an die Implementierung soll der Code-Editor nun evaluiert werden. Dazu soll eine vergleichende Nutzerstudie zwischen dem in dieser Arbeit entwickelten SEE-Editor und einem normalen Editor, dem Windows-Editor, durchgeführt werden. Dadurch soll die Forschungsfrage „Wie unterscheidet sich die Effektivität kollaborativen Arbeitens bei einem in SEE eingebetteten, mehrbenutzerfähigen Editor von der Effektivität der Nutzung eines normalen Texteditors?“ zu beantworten.

Dazu werden zunächst die Grundlagen dieser Evaluation erläutert, darauf aufbauend wird dann die Auswahl sowie der Aufbau des Fragebogens beschrieben. Im Anschluss wird die Aufgabe definiert und die Durchführung der Pilotstudie, sowie die Erfahrung mit der Durchführung der Studie beschrieben. Zuletzt werden die Ergebnisse der Evaluation ausgewertet und es werden einige Bedrohungen zur Validität der Studie genannt und untersucht.

6.1 Grundlagen der Evaluation

Zunächst werden die Grundlagen, auf denen die Evaluation aufbaut, erläutert. Dazu werden zunächst einige Anforderungen an die Studie definiert, im Anschluss wird dann beschrieben, welche Daten erhoben werden sollen und zum Schluss wird die Aufteilung der Probanden erklärt.

6.1.1 Anforderungen

Damit die Durchführung der Studie möglichst reibungslos klappt, werden im Folgenden einige Anforderungen an die Studie beschrieben.

- Die Studie sollte online durchführbar sein, zum einen aufgrund der anhaltenden Covid-19-Pandemie, zum anderen, da es deutlich einfacher ist, verschiedene Probanden zeitgleich online zu organisieren.

- Es sollte möglichst einfach für die Teilnehmer sein, an der Evaluation teilzunehmen. Deswegen sollte der Fragebogen im Browser machbar sein und die benötigte Videokonferenz sollte auch ohne das Installieren von Software durchführbar sein. Nur SEE selbst musste heruntergeladen werden, da dies schlecht im Browser ausführbar ist.

- Es sollte bei dem Fragebogen möglich sein, dass dieser unterbrochen und zu einem späteren Zeitpunkt fortgeführt werden kann, falls die Gruppe

aus einem Grund unterbrochen wird oder der Browser aus Versehen beendet wird, bevor der Fragebogen fertig ausgefüllt ist.

- Die Tools für die Studie sollten kostenlos sein.

- Es sollten, wenn möglich, Tools der Uni-Bremen verwendet werden, da dort das Level des Datenschutzes jedem Teilnehmer bekannt sein sollte und er sich von der Verwendung nicht abschrecken lässt.

6.1.2 Erhobene Metriken

Um die Studie auswerten zu können, werden folgende Metriken erhoben.

- Die erste Metrik ist die Dauer der Aufgabe. Der Grund, wieso diese Metrik erhoben wird, ist, Entwickler wollen möglichst effizient arbeiten können.

  - **Nullhypothese:**
    Die Aufgabe mit dem normalen Editor wird mindestens gleich schnell gelöst, wie die mit dem SEE-Editor.

  - **Alternativhypothese:**
    Die Aufgabe mit dem SEE-Editor wird schneller gelöst als die mit dem normalen Editor.

- Als zweite Metrik wird die Korrektheit betrachtet. Denn wenn Entwickler zusammen bestimmte Aufgaben bearbeiten oder Metriken erheben wollen, sollte am Ende jeder Entwickler über die gleichen Informationen verfügen, da sonst Fehlschlüsse oder auch falsche Aktionen folgen könnten.

  - **Nullhypothese:**
    Die Aufgabe mit dem normalen Editor wird mindestens gleich korrekt gelöst wie die mit dem SEE-Editor.

  - **Alternativhypothese:**
    Die Aufgabe mit dem SEE-Editor wird korrekter gelöst als die mit dem normalen Editor.

- Zuletzt wird noch die Benutzbarkeit erfasst. Da die Benutzbarkeit eine subjektive Metrik ist, soll diese mithilfe eines Fragebogens nach jeder Aufgabe erfasst werden.

- **Nullhypothese:**
  Der Nutzer empfindet es mindestens als genauso angenehm, die Aufgabe mit dem normalen Editor zu lösen als die mit dem SEE-Editor.

- **Alternativhypothese:** Der Nutzer empfindet es als angenehmer, die Aufgabe mit dem SEE-Editor zu lösen, als die mit dem normalen Editor.

6.1.3 Zielgruppe

Die Zielgruppe von SEE selbst sind eher Softwareentwickler und Softwarearchitekten. Von dieser Zielgruppe wird für die Evaluation allerdings abgewichen.
Dies wird gemacht, da es einerseits relativ schwierig ist, genug Gruppen zu sammeln — da die Evaluation immer in Dreiergruppen durchgeführt wird, werden für nur zehn Datenpunkte schon 30 Probanden benötigt. Andererseits, da der Anwendungsfall, Informationen aus einer Visualisierung zu entnehmen und diese in einem Editor zu sammeln, auch bei anderen Anwendungen denkbar ist. Außerdem wird für die Durchführung auch kein spezielles Wissen vorausgesetzt, außerhalb des Umgangs mit einem Computer und der Lese- und Schreibfähigkeiten.
Deswegen wurden folgende Segmente von Teilnehmern geplant:

- Personen, die an SEE mitgewirkt haben.

- Personen, die Informatik, Wirtschaftsinformatik oder Systemsengineering studieren, aber nicht an SEE gearbeitet haben.

- Personen, die studieren, aber nicht zu den oben genannten Gruppen gehören.

- Personen, die nicht studieren.

6.2 Auswahl des Fragebogens

Nachdem die Grundlagen feststehen, wird nun der Fragebogen konzeptioniert. Es werden zunächst ein paar einleitende Fragen gewählt. Danach folgt die Auswahl eines standardisierten Fragebogens, um die Benutzbarkeit zu erfassen und zusätzlich werden noch drei Fragen direkt zur Aufgabe gestellt. Am Ende der Aufgabe wird noch das Resultat dieser abgefragt und die zur Lösung benötigte Zeit.

6.2.1 Einleitende Fragen

Zu Beginn der Studie werden ein paar demografische Daten abgefragt, um eventuelle Auffälligkeiten bei der Auswertung später genauer analysieren zu können. In diesem Fragebogen erfasst werden: Geschlecht, Alter, Bildungsabschluss. Zusätzlich wurden noch Fragen zum Vorwissen gestellt, konkret nach der Erfahrung mit Softwareentwicklung, SEE selbst und Computerspielen, konkret nach *First-/ Third-Person PC-Spielen*, gefragt.

First-/ Third-Person PC-Spiele: Computerspiele, bei denen der Spieler aus der Perspektive der Spielfigur oder über dessen Schulter die Spielwelt sehen kann.

Bestehende Erfahrung mit Softwareentwicklung könnte dazu führen, dass die Teilnehmer die Aufgabe schneller verstehen, da sie an bestimmte Begriffe und Visualisierungen schon gewöhnt sind, die in der Anleitung vorkommen.
Bei der Erfahrung mit SEE geht es vor allem um den Umgang mit den Features von SEE. Wenn ein Teilnehmer zum Beispiel an SEE mitentwickelt hat, sollte dieser im Umgang mit dieser speziellen virtuellen Welt, der Visualisierung und dem Handling besser vertraut sein und effektiver arbeiten können. Hingegen ein Teilnehmer, der SEE noch nie vorher gesehen hat, muss sich vermutlich an einige Eigenschaften erst gewöhnen.
Auch die Erfahrung mit Computerspielen kann eine Rolle spielen, da SEE einem solchen in Teilen ähnelt. Diese Ähnlichkeit zeigt sich zum Beispiel in der Steuerung und der Ego-Perspektive, welche auch von einigen anderen Spielen verwendet wird. Dadurch kann es sein, dass erfahrene Spieler effektiver in der virtuellen Welt zurechtkommen.

6.2.2 Fragen zur Benutzbarkeit

Um einen Eindruck zur Benutzbarkeit zu bekommen, werden hierzu gezielt Fragen gestellt. Damit das Ergebnis aussagekräftig ist, soll ein Fragebogen verwendet werden, welcher bereits standardisiert ist und von anderen verwendet wurde. Dazu wird in einer Recherche unter anderem eine Liste des Fraunhofer-Instituts mit verschiedenen Fragebögen zur Benutzbarkeit gefunden. [Fra]
Wichtig bei der Auswahl war zudem auch, dass der Fragebogen nicht zu umfangreich ist, da die Bearbeitung der Aufgabe an sich schon ca. 20 Minuten dauert. Zusätzlich sind noch ca. 15 Minuten für eine Einführung und den Aufbau des Systems mit allen Teilnehmern einzuplanen. Da die Fragen zweimal beantwortet werden müssen, einmal für den SEE-Editor und einmal für den Windows-Editor, wird versucht einen Fragebogen mit fünf bis fünfzehn Fragen zu finden.

In der Liste des Fraunhofer-Instituts wurden insgesamt achtzehn Fragebögen vorgestellt. Sechs davon kamen von vornherein nicht infrage, da sie sich auf Webseiten bezogen. Bei den restlichen zwölf wurde zwischen Fragebögen im

Anschluss an eine Produktstudie, auch post-study genannt und in Anschluss an eine Aufgabe, auch post-task genannt, unterschieden.
Bei der Evaluation werden gewissermaßen zwei komplette Systeme miteinander verglichen. Zum einen das System, in welchem nur SEE mit dem Sprachchat verwendet wird, um die Informationen auszutauschen. Zum anderen das System, in welchem SEE in Kombination mit einem externen Editor und einer kompletten Onlinekonferenz-Lösung (Sprachchat, Videochat, Textchat, Bildschirm teilen) verwendet wird.
Zusätzlich wird dabei nicht mehr als eine Aufgabe pro System erledigt. Daher werden die post-task Fragebögen nicht weiter beachtet, da zur Aufgabe konkret eigene Fragen gestellt werden, um die einzelnen Aspekte der Aufgabe besser hervorzuheben.
Nach dieser ersten groben Selektion bleiben noch sieben post-study Fragebögen über. Näher betrachtet werden folgende Fragebögen.

- **AttrakDiff**, ein Fragebogen, welcher die *pragmatische Qualität* und *hedonische Qualität* sowie die Attraktivität eines Tools erhebt. Dieser Fragebogen fällt allerdings aufgrund des Umfangs von 23 Fragen heraus.

pragmatische Qualität: Bezeichnet, wie nützlich respektive praktisch in der Nutzung, ein Tool ist.[Mül10]

hedonische Qualität: Bezeichnet, welche Emotionen ein Tool hervorruft.[Mül10]

- **Post-Study System Usability Questionaire (PSSUQ)**, zur Erfassung der Zufriedenheit der Nutzer mit dem Tool. Dieser Fragebogen umfasst sechzehn Fragen, was zwar länger ist als ursprünglich geplant, aber aufgrund der geringen Differenz trotzdem noch in die engere Auswahl kommt.

- **Questionaire for User Interface Satisfaction**, dieser Fragebogen erfasst die Zufriedenheit von Nutzern mit einer Mensch-Maschinenschnittstelle. In der umfangreichen Version umfasst dieser Fragebogen 90 Fragen und in der kurzen 20 Hauptfragen und fünf Zusatzfragen. Das ist deutlich zu lang, daher ist dieser Fragebogen nicht in der engeren Auswahl.

- **Software Usability Measurement Inventory**, „Erfasst die Nutzungsqualität der Software aus Sicht des Benutzers“[Fra]. Auch dieser Fragebogen ist mit 75 Fragen deutlich zu lang und wird nicht näher betrachtet.

- ***System Usability Scale*** **(SUS)**, erfasst die Nutzerzufriedenheit in Bezug auf die Benutzbarkeit eines Tools. Dieser Fragebogen ist mit zehn Fragen angenehm kurz und wird näher betrachtet.

SUS: Ein validierter Fragebogen zur Erfassung der Benutzbarkeit von Software.

- **Usability Metric for user experience**, ist angelehnt an den SUS, ist allerdings als deutlich kürzere Alternative gedacht. Dieser Fragebogen ist mit

vier Fragen sehr kurz und wird deswegen nicht näher betrachtet.

- **User experience questionaire**, dieser Fragebogen „Erfasst die klassischen Usability-(Effizienz, Durchschaubarkeit, Verlässlichkeit) und User Experience-Aspekte(Attraktivität, Stimulation, Originalität)“[Fra]. Dieser Fragebogen ist allerdings 26 Fragen lang, was für diese Studie deutlich zu lang ist und wird deswegen nicht näher betrachtet.

In der engeren Auswahl sind lediglich zwei Fragebögen angekommen. Der PSSUQ und der SUS. Von dem PSSUQ wurde keine validierte deutsche Version gefunden. Der SUS hingegen wurde durch ein Crowdsourcing-Projekt von Wolfgang Reinhardt von verschiedenen Usability Experten auf Deutsch übersetzt und zur Validierung von verschiedenen Muttersprachlern britischen und amerikanischen Englischs wieder ins Englische übersetzt. Damit sollte sichergestellt werden, dass die Bedeutung der Fragen gleich geblieben ist. [Ber16] Bei der genannten Quelle, „System Usability Scale - jetzt auch auf Deutsch.“von Bernard Rummel, gibt es zusätzlich die deutschsprachige Version als Download, anhand derer der Fragebogen dann erstellt wird. Deswegen und aufgrund der Kompaktheit durch die nur zehn statt sechzehn Fragen, wurde sich für den SUS entschieden.

Zusätzlich zum SUS wurden noch drei konkrete Fragen zur Aufgabe erstellt und ein Feld für Anmerkungen im Fragebogen eingebaut.
Die drei separaten Fragen lauten:

- Ich finde es einfach, mit diesem System mit mehreren Leuten Informationen zusammenzutragen.
- Ich finde es einfach, mit diesem System mit mehreren Leuten Informationen zu sortieren.
- Ich finde, es war einfach, die gegebene Aufgabe mit dem System zu lösen.

Die Idee hinter diesen zusätzlichen Fragen ist es, zusätzlich zur Benutzbarkeit direkt zum System in Bezug zur Aufgabe Feedback zu bekommen, um eventuelle Schwachstellen zu identifizieren.
Die Antwortmöglichkeiten wurden hier so wie beim SUS gewählt, um den Übergang zwischen den Fragen zu erleichtern. Als Letztes wird in einem offenen Eingabefeld noch nach weiteren Anmerkungen zum System gefragt, da-

mit Punkte, die vom SUS oder den Zusatzfragen nicht abgedeckt wurden, auch angebracht werden können.

6.2.3 Korrektheit und Zeit

Um die Zeit pro Gruppe zu erfassen, wird diese im Fragebogen nach dem Beenden eines Teils vor dem SUS abgefragt. Diese wurde vom Experiment Leiter erfasst und den Teilnehmern danach durchgegeben.

Zusätzlich sollte auch die Korrektheit gemessen werden. Dazu wurde nach der Zeit noch nach dem Lösungstext gefragt. In dieses Eingabefeld sollten die Teilnehmer jeweils den Inhalt ihres in der Aufgabe genutzten Editors in den Fragebogen kopieren, damit dieser später innerhalb der Gruppen verglichen werden kann.

6.2.4 Erstellung des Fragebogens

Um den Fragebogen zu erstellen, wird ein Online-Tool verwendet. Dafür standen zunächst zahlreiche Möglichkeiten im Raum. Nach ein wenig Recherche stellte sich heraus, dass die Universität Bremen über eine SoSci-Lizenz verfügt und dieser Service auch auf den Servern der Universität Bremen gehostet wird.
Da für die Evaluation möglichst Werkzeuge der Universität Bremen genutzt werden sollen, wurde sich für diesen Service entschieden. Des Weiteren ist dieser kostenlos und es ist möglich, Links zu den Fragebögen mit einer ID auszustatten. Dies ermöglicht es zum einen bei der Auswertung die Ergebnisse nach Gruppen auszuwerten. Zum anderen ermöglicht es, dass die Nutzer den Fragebogen auch unterbrechen können, da die Antworten direkt gespeichert werden und mithilfe der ID auch wieder aufgerufen werden können.
Wie der Fragebogen selbst aussieht, ist dem Anhang *Fragebogen* zu entnehmen.

6.3 Aufgabenstellung

Um eine passende Aufgabe zu finden, wird zunächst überlegt, mit was für einer Art von Aufgabe die Forschungsfrage validiert werden kann.
Die Idee ist, dass in der Evaluation der Editor als Notizzettel genutzt wird, um, wie in Unterunterabschnitt 6.1.3 erwähnt, die Zielgruppe möglichst groß zuhalten. Die Aufgabe ist also, Informationen zu sammeln und diese dann auf dem Notizzettel festzuhalten. Dies könnte in der Praxis vorkommen, wenn sich mehrere Entwickler zusammen eine größere Software anschauen und zum Beispiel die Klassen mit den meisten Codezeilen aus dem Projekt suchen wollen, um später darüber zusprechen. Ein weiteres Szenario wäre auch, dass sich die Entwickler in SEE treffen und To-dos erarbeiten, aufschreiben und nach

Wichtigkeit sortieren.

Zuerst war die Idee, dass im ersten Durchlauf, die Dateien mit der größten LOC und im Vergleichsdurchlauf dann die mit der größten McCabe Komplexität, gefunden werden sollen.

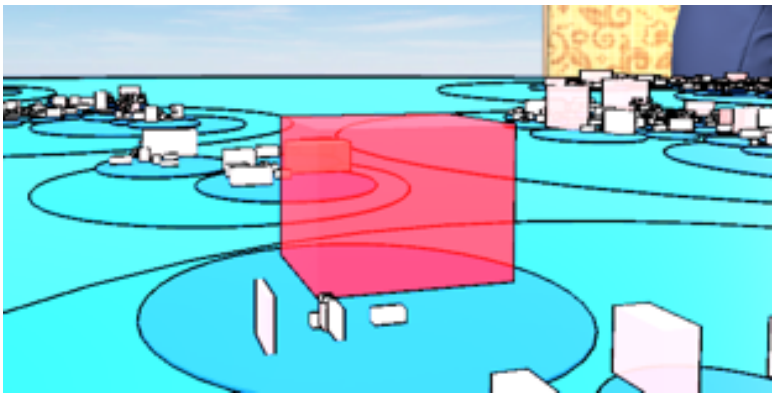

Abbildung 14: Die Farbe der Gebäude stellt die McCabe Komplexität dar und die Höhe die LOC.

Dabei ist aufgefallen, dass zum einen die unterschiedlichen Aufgaben auch für Unterschiede in der Schwierigkeit zwischen den beiden Durchläufen sorgen könnte, da die LOC durch die Höhe des Knotens und die McCabe Komplexität durch die Farbe (Rot) dargestellt wird. Die Vermutung ist, dass es deutlich einfacher ist, die Farbe zu erkennen als die Höhe. Zum anderen ist es schwierig, in den Notizen die Dateien nach Farbe zuordnen, da es für den ungeübten Betrachter nicht eindeutig erkennbar ist, welcher RGB-Wert hinter der Knotenfarbe steht. Man könnte zwar die Komplexität auch als Zahl im Knotennamen mit angeben, allerdings bedeutet das dann auch wieder zusätzlichen Programmieraufwand und müsste auch für die LOC angepasst werden. Das letzte und vermutlich auch wichtigste Argument ist aber, dass es in der gesamten Stadt wenig rote Knoten gibt und somit nicht so viele zur Auswahl stehen. Deswegen wird sich dafür entschieden, dass nur die LOC gesucht wird und im ersten und zweiten Teil unterschiedliche Bereiche der Software-City untersucht werden.

Die Aufgabe sieht nun vor, dass jeder Teilnehmer einen inneren Knoten zugewiesen bekommt, in dem sich noch weiter Kindknoten befinden. Der Teilnehmer sollte jetzt die fünf größten Kindknoten finden und in diesen jeweils den höchsten Blattknoten identifizieren.

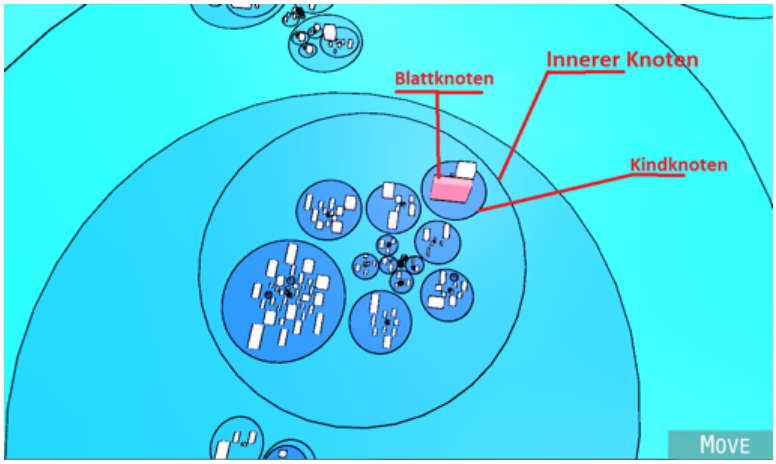

Abbildung 15: Beschriftung Knoten.

Die gefundenen Knoten werden dann von den Teilnehmern im Editor festgehalten. Dabei wird dann auch die LOC mit angegeben. Das Ziel ist es, dass in der Datei am Ende alle fünfzehn Blattknoten sortiert nach der LOC stehen sollen. Um die Forschungsfrage zu klären, wurde das Experiment in zwei Aufgaben geteilt.

- Die Lösung der Aufgabe mit dem SEE-Editor.
- Die Lösung der Aufgabe mit dem Windows-Editor. Hierbei ist wichtig, dass jeder Spieler am Ende die gleichen Informationen in seinem lokalen Editor stehen haben soll. Wie die Teilnehmer die Ergebnisse synchronisieren, können sie frei wählen. Da für diese Studie neben SEE noch ein Videokonferenzsystem benutzt wird, können die üblichen Mittel benutzt werden, wie etwa das Screensharing oder der Chat.

Nach der Durchführung einer Pilotstudie ist allerdings aufgefallen, dass fünf Knoten zu viel sind und es besser wäre, diese zu reduzieren. Deswegen mussten die Probanden pro Aufgabe nur noch drei Knoten finden.
Nach der Lösung der Aufgaben sollte die Dateien mit der Aufgabe, wie auf Abbildung 16 zu sehen ausgefüllt sein:

Abbildung 16: Eine Beispiellösung für eine Aufgabe.

In der Datei wurde auch noch die Aufgabe wiederholt und jeweils für Spieler eins bis drei ein Bereich an Zeilen festgelegt, in dem der Spieler seine Ergebnisse festhalten sollte. Die Datei wurde im internen SEE-Editor genauso gestaltet wie im Windows-Editor und das Ergebnis sollte bei beiden Aufgaben gleich aussehen, es wurden unterschiedliche Bereiche bei beiden Aufgaben gewählt, also die Namen der Knoten waren bei den Endprodukten unterschiedlich.
Es sollte also zuerst jeder Spieler seine drei größten Dateien aufschreiben — hier war die Sortierung noch nicht unbedingt gefordert, konnte aber bei dem zweiten Teil der Aufgabe helfen. Im zweiten Teil der Aufgabe sollte dann eine sogenannte *Top neun* Liste erstellt werden, damit ist eine nach LOC absteigend sortierte Liste der Einträge der drei Teilnehmer gemeint. Um die Namen, die teilweise auch etwas länger sein konnten, nicht abtippen zu müssen, wurde empfohlen, unten eine Abkürzung zu verwenden. (Siehe Abbildung 16) Die empfohlene Abkürzung sieht wie folgt aus: $Px.y$. Dabei steht P als Abkürzung für Player, x stellte die Spielernummer dar und y den Eintrag eins bis drei des Spielers x.

Der Hauptunterschied zwischen den beiden Varianten der Aufgabe ist, dass der Windows-Editor nicht automatisch die Ergebnisse der anderen Spieler mit anzeigt und somit ein größerer Aufwand und eine potenzielle Fehlerquelle entsteht, da die Teilnehmer ihre Ergebnisse untereinander abgleichen müssen. Da alle Spieler während der Durchführung in einer Big Blue Button (BBB) Konferenz waren, durften sie den Funktionsumfang des Konferenzsystems nutzen. Eine Ausnahme war die *geteilte Notizen* Funktion von BBB, da diese ein Kollaborativer-Editor ist.
Bei dem Teil mit dem in SEE integrierten Editor war zu erwarten, dass es we-

niger Fehler beim Informationsaustausch gibt und dieser schneller geschehen sollte, da die Teilnehmer nicht mehr aktiv Informationen austauschen müssen.

6.4 Die Wahl der Knoten

Die Auswahl der zu bearbeitenden Knoten wurde mehrfach überarbeitet und überdacht. Wichtig war zum einen, dass die Knoten nicht zu dicht beieinanderliegen, sodass sich die Teilnehmer nicht gegenseitig behindern, andererseits mussten die Knoten auch eine angemessene Anzahl an Kindknoten haben. Ist diese zu gering, gibt es nicht genug Knoten im Bereich, um die Aufgabe zu erfüllen, ist sie zu groß, ist es schwierig für die Teilnehmer den Bereich zu überblicken und die richtigen Knoten zu finden.
Des Weiteren musste auch darauf geachtet werden, dass die Höhe der Blattknoten angemessen unterschiedlich ist oder aber nicht zu viele Blattknoten zur Auswahl stehen, sodass diese einfach überprüft werden können. Zudem müssen die Knoten von beiden Durchgängen vergleichbar sein, damit die Wahl der Knoten keinen Einfluss auf die Ergebnisse hat.

Die Knoten, die letztlich für diese Aufgabe gewählt worden sind: *IO, Evolution, Tools* für den Windows-Editor und *EdgeLayouts, UI3D, CameraPaths* für den SEE-Editor.
Hierbei wurde auch darauf geachtet, dass die Knoten pro Teilnehmer bei beiden Teilen nah beieinanderliegen, damit sie schneller gefunden werden können, um keine Zeit zu verschwenden.

6.5 Pilotstudie

Nachdem die Vorbereitungen abgeschlossen sind, wird eine Pilotstudie angestrebt. Diese Pilotstudie musste jedoch mehrmals abgebrochen werden, da dabei immer wieder Probleme aufgetreten sind. Im Folgenden werden diese Probleme und ihre Lösungen kurz beschrieben.

6.5.1 Netzwerk Überlastung

Es gab ein technisches Problem im Programmcode des SEE-Servers, dass das Verbinden von mehr als zwei Spielern unmöglich machte. Durch die ständige Synchronisierung der Spieler-Köpfe wurde der Server zu stark ausgelastet.

Abbildung 17: Die Spieler-Köpfe im Mehrbenutzermodus.

Diese Köpfe präsentieren je einen Teilnehmer und dessen Position in der SEE-Welt. Die Position der Köpfe muss im Netzwerk übertragen werden, dies geschah in der alten Version alle *100 ms*. Um die Netzwerkbelastung zu verringern, wurde dies geändert, die Position wird nur noch gesendet, wenn sich der Kopf bewegt hat und seit der letzten Bewegung mindestens *300 ms* vergangen sind.
Diese Änderung brachte ein Problem mit sich — der Kopf blieb nach der letzten Bewegung nicht stehen, sondern flog weiter geradeaus. Dies konnte durch eine Änderung in der Positionsberechnung von einer unbeschränkten zu einer beschränkten Interpolation gelöst werden.

6.5.2 Zoomen und verschieben der Stadt

Als Nächstes fiel auf, dass wenn ein Nutzer die Stadt vergrößert und dann den gezeigten Ausschnitt verschiebt, (siehe Abbildung 18) die Stadt bei den anderen Nutzern verschwindet.

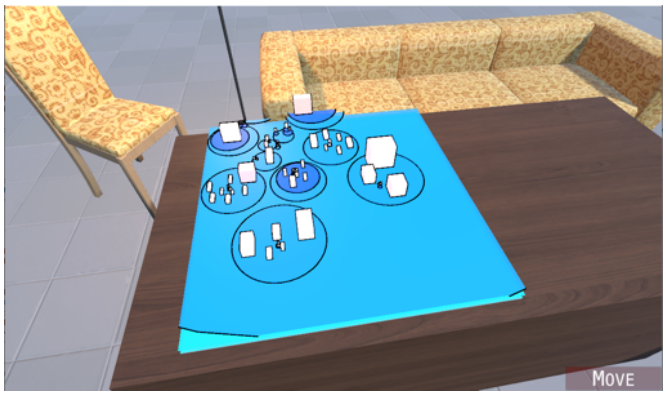

Abbildung 18: SEE-City vergrößert.

Dies geschah, da das Zoomen nicht synchronisiert wird, aber die Positionsänderung beim Verschieben. Da das Zoomen für die Studie nützlich ist, während auf das Verschieben verzichtet werden kann, wurde dieses Problem ignoriert und den Teilnehmern angewiesen, die Stadt nicht zu verschieben.

6.5.3 Editor Synchronisierungsprobleme

Es ist aufgefallen, dass die Synchronisierung des Textes fehlerhaft funktioniert, wenn ein Nutzer *zu schnell tippt*. Durch das schnelle Tippen wird von der $inputString$ Methode ein String ausgegeben, welcher mehr als ein Zeichen enthält. Dadurch geriet die Position durcheinander, es konnte allerdings durch ein Anpassen der Cursor-Positionsberechnung gelöst werden. Diese musste lediglich um die Länge des Strings nach links verschoben werden.

6.5.4 Netzwerkauslastung durch Code-Windows

Als letztes Problem ist aufgefallen, dass das Öffnen von mehreren Code-Windows eine zu große Netzwerklast für den Server darstellen kann. Da nach dem Öffnen die teilweise sehr großen Dateien über das Netzwerk versendet werden müssen. Da dies nicht ohne großen Aufwand reduzierbar ist, wurde auf das Öffnen von anderen Dateien in der Studie verzichtet. Damit die Teilnehmer trotzdem die LOC notieren können, wird diese nach einer Änderung nun hinter dem Knotennamen angezeigt. (Siehe Abbildung 19)

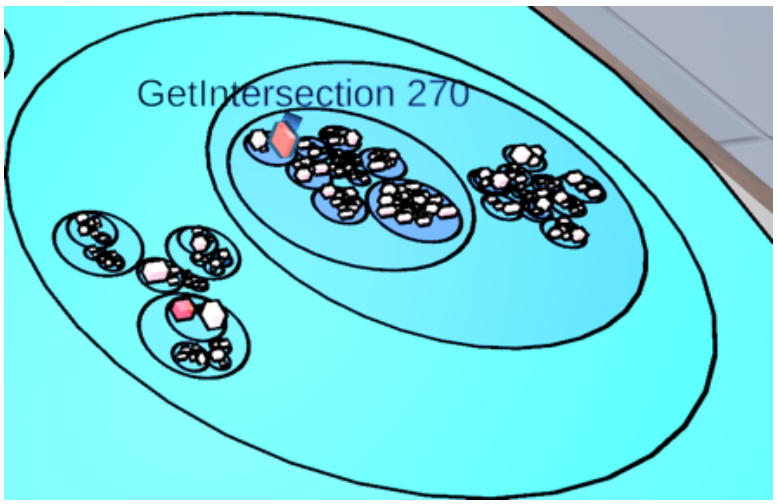

Abbildung 19: Die Anzahl der Code Zeilen im Knoten Namen.

6.6 Planung und Durchführung der Studie

Um die Studie nach den Pilot-Erfahrungen erfolgreich mit Probanden durchzuführen, musste nun also ein genauer Plan erstellt werden, um nicht dieselben Fehler noch einmal zu machen. In diesem Teil werden zunächst die genaue

Planung der Studie erläutert. Zum Schluss wird dann von den Erfahrungen der Durchführung berichtet.

6.6.1 Planung der Studie

Auf Grundlage dieser Erfahrungen im Rahmen der Pilotstudie war es besonders wichtig, noch einmal das genaue Vorgehen bei der Durchführung der Studie zu strukturieren. Zur Erklärung von SEE und dessen Verwendung wurde zunächst eine Präsentation erstellt, die jeder Proband vor der Durchführung der Studie nutzen sollte, um den Einstieg zu erleichtern. Diese Präsentation ist auch im Anhang unter *Präsentation.pdf* zu finden. Der Inhalt der Präsentation umfasste folgende Punkte:

- Ein Glossar für die Begriffe, welche im Rahmen der Studie benötigt wurden. (Abbildung 15)
- Abbildungen, zur Erklärung der Suche nach Knoten und Komponenten, sowie der Notation der Ergebnisse. (Abbildung 16 und Abbildung 19)
- Eine annotierte Version der Softwarecity zur besseren Findung der zu suchenden Komponenten

Damit jeder Teilnehmer eine SEE Anwendung benutzen kann, wurde diese über einen Cloud-Service mithilfe eines Downloadlinks bereitgestellt. Die hochgeladene SEE-Applikation musste allerdings an jedem Tag ausgetauscht werden, da sich die dort eingetragenen IP-Adressen des Universitäts-VPNs täglich änderten.
Als Videokonferenz-System zum Vergleich mit dem integrierten Editor wurde Big-Blue-Button genutzt, da es ohne eine Installation im Browser verwendbar ist und somit eine einfache Nutzung für die Probanden ermöglicht.
Die Teilnehmer werden über verschiedene Kanäle akquiriert, so werden einige persönlich und andere über Gruppen angeschrieben. Die Einteilung in die Dreiergruppen erfolgt manuell, basierend auf den Angaben der Teilnehmer, wer wann Zeit hat.
Die Dreiergruppen werden in zwei Partitionen geteilt, A und B Gruppen. Die Einteilung erfolgt abwechselnd, Gruppe-1 gehört zur A Partition, Gruppe-2 zur B Partition usw. Die Partitionen unterscheiden sich in der Reihenfolge, in der die Aufgaben bewältigt werden. A benutzt zuerst den SEE-Editor und dann den Windows-Editor, bei Partition B ist die Reihenfolge dann exakt umgekehrt.
Dadurch sollte sichergestellt werden, dass die Reihenfolge der Testobjekte keine Rolle bei der Auswertung einnimmt. Da die Teilnehmer nach dem ersten Teil schon Erfahrung mit der Aufgabe und SEE gesammelt haben.
Falls Teilnehmer Probleme haben sollten, SEE auszuführen, werden zwei betriebsbereite Systeme vorgehalten, diese können dann über eine Fernsteuerung von den Teilnehmern benutzt werden.

6.6.2 Erfahrungen mit der Studie

Die Studie konnte erfolgreich durchgeführt werden. Insgesamt haben 24 Probanden in acht Gruppen die Studie erfolgreich absolviert. Die A und B Partitionen waren gleichmäßig mit je vier Dreiergruppen vertreten.

Probleme der Studie Bei der zweiten Dreiergruppe gab es ein Problem. Durch eine zu langsame Internetanbindung eines Probanden wurde der Inhalt des SEE-Editors nicht richtig synchronisiert. Dadurch wurde der Inhalt verdoppelt. (Siehe Abbildung 20.)

Abbildung 20: Doppelter Text im Code-Window.

Um dieses Problem zu lösen, wurde eine experimentelle Version von SEE integriert. Diese Version verbesserte die Netzwerkleistung, brachte allerdings auch zwei Bugs mit sich. Einerseits wurden die Teilnehmer beim Vergrößern der Software-Stadt durch diese verschoben, andererseits war auch während des Tippens eine Bewegung möglich und es wurde ein Chat geöffnet, wenn bestimmte Buchstaben gedrückt wurden.
Diese neuen Probleme konnten allerdings erfolgreich identifiziert und gelöst werden. Dies geschah allerdings erst nach dem insgesamt vier Gruppen die Studie absolviert hatten.
Mit dieser Änderung wurden auch die Köpfe (Siehe Abbildung 17) durch Menschen ähnlichere Avatare ersetzt (Siehe Abbildung 21). Zusätzlich konnten die Netzwerkeinstellungen nun beim Programmstart angepasst werden. Dies erleichterte die Durchführung deutlich, da dadurch nicht mehr vor jedem Durchlauf eine neue Version erstellt und hochgeladen werden musste.

Abbildung 21: Repräsentation eines Spielers als Menschenähnliches Avatar.

Rückmeldungen der Teilnehmer Die Evaluationen dauerten im Schnitt ca. eine Stunde. Diese Dauer kam primär durch einige Probleme mit der Vorbereitung zustande. Bis auf wenige Ausnahmen klappte die Durchführung der Studie nicht beim ersten Versuch. Die dabei auftretenden Probleme waren vielfältig, konnten allerdings alle nach einigen Neustarts der Anwendung oder dem Ausweichen auf einen ferngesteuerten Klienten behoben werden.
Trotz dieser Unannehmlichkeiten waren die Rückmeldungen überwiegend positiv. Zwar merkten fast alle Probanden an, dass bis zur Produktreife noch einige Kleinigkeiten im Editor sowie auch in SEE selbst verbessert werden sollten. Trotzdem waren die meisten Teilnehmer davon überzeugt, dass die Idee hinter SEE sowie auch dem Editor gut sind und den Umgang mit Software erleichtern könnten.

6.7 Auswertung

Nach Abschluss der Benutzerstudie wird die Auswertung dieser durchgeführt. Dies geschieht in vier Abschnitten. Zunächst werden die demografischen Daten ausgewertet. Im nächsten Schritt soll die Dauer und Korrektheit der durchgeführten Aufgaben bestimmt werden und daraus Rückschlüsse auf die Effektivität des jeweiligen Testobjekts gezogen werden. Abschließend wir dann der SUS und die eigenen Fragen beider Objekte ausgewertet und miteinander verglichen.

6.7.1 Demografie

Im Bereich der Demografie werden nun die in Unterunterabschnitt 6.2.1 genannten Aspekte beschrieben, und im Folgenden schrittweise ausgewertet.

- Geschlecht
- Alter

- Bildungsgrad
- Erfahrung mit Softwareentwicklung
- Erfahrung mit SEE
- Erfahrung mit First-/ Third-Person PC-Spiele

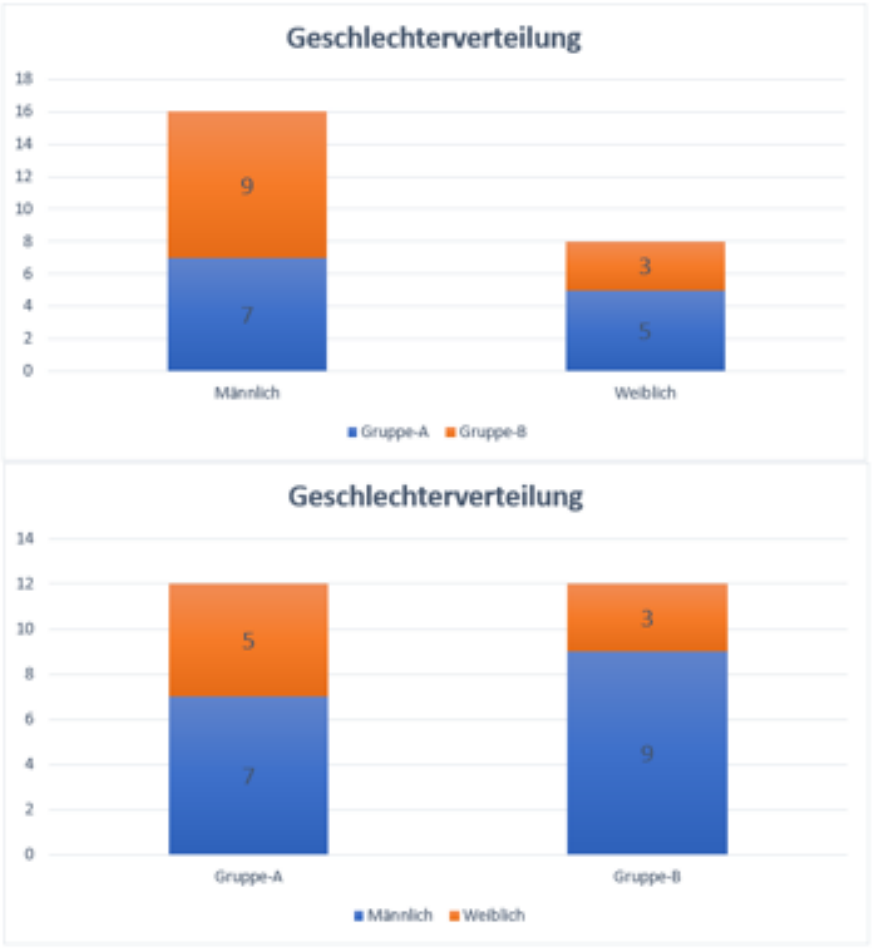

Abbildung 22: Geschlechterverteilung der Probanden.

Geschlechterverteilung Wie dem Säulendiagramm zu entnehmen, ist der überwiegende Teil der Probanden männlich (66%) und nur ein kleiner Teil weiblich (33%).
Zudem fällt bei der Geschlechterverteilung innerhalb der Gruppen auf, dass es einen deutlichen Unterschied gibt, in Gruppe-A sind $41,6\%$ weiblich und in Gruppe-B nur 25%. Der Einfluss dieser Ungleichverteilung ist aber vermutlich eher gering, da die Teilnehmer in ihren Dreiergruppen kaum nach Geschlecht aufgeteilt waren. So gibt es unter allen Gruppen nur zwei, die nur aus einem Geschlecht bestehen, beide sind rein männliche Gruppen. Im Vergleich mit den anderen Gruppen sind diese aber weder durch besonders gute, noch besonders schlechte Resultate aufgefallen.

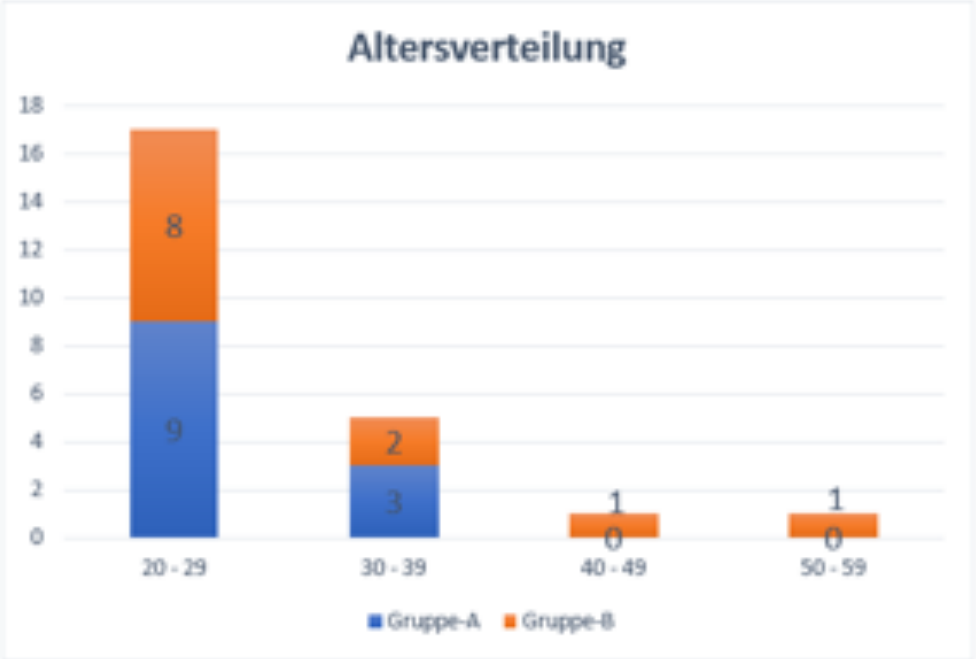

Abbildung 23: Die Altersverteilung der Probanden.

Altersverteilung [9] [10] Bei der Altersverteilung sind beide Gruppen deutlich ausgewogener. Insgesamt sind $70,8\%$ der Probanden unter 30, dies teilt sich in 75% bei Gruppe-A und $66,6\%$ bei Gruppe-B auf. Wie der Grafik zu entnehmen, ist Gruppe-B im Durchschnitt allerdings ein wenig älter $25,8$ zu $22,5$ Jahren [11].

[9] Die Darstellung wurde auf 10 Jahre erweitert, im Fragebogen waren es ursprünglich fünf Jahresgruppen, es war jedoch kein Mehrwert für die genauere Darstellung erkennbar.

[10] Nicht gewählte Optionen wurde aus dieser und folgenden Grafiken entfernt, um die Übersichtlichkeit zu steigern.

[11] Zur Durchschnittsberechnung wurde immer die untere Grenze des Altersbereichs benutzt (20,30,40,50 Jahre)

Abbildung 24: Der Bildungsgrad der Probanden.

Bildungsgrad Die meisten Teilnehmer haben als höchsten Bildungsabschluss Abitur (54%). Dabei fällt auf, dass bei Gruppe-A eine höhere Bildung vorliegt, 100% der Teilnehmer haben mindestens Abitur. Bei Gruppe-B haben $33,3\%$ der Teilnehmer lediglich einen Realschulabschluss. Bei Gruppe-A hingegen haben $33,3\%$ der Teilnehmer einen Hochschulabschluss, Bachelor oder Master sind hierbei gleich verteilt.
Bei einem Blick auf die Rohdaten fällt auf, dass die beiden Dreiergruppen, welche jeweils zwei Teilnehmer mit Realschulabschluss und einen Abitur-Abschluss haben, bei dem nicht Windows-Editor in Bezug auf die benötigte Zeit mit deutlichem Abstand den letzten und vorletzten Platz belegten.

Abbildung 25: Erfahrung mit Softwareentwicklung.

Erfahrung mit Softwareentwicklung Die Auswertung der Erfahrung mit Softwareentwicklung zeigt, dass $37,5\%$ der Teilnehmer mindestens drei Jahre Erfahrungen im Bereich der Softwareentwicklung gesammelt haben. $58,3\%$ der Teilnehmer hatten zumindest regelmäßig Erfahrung mit Softwareentwicklung, lediglich $41,6\%$ der Probanden hatten nur gelegentlich oder keine näheren Berührungspunkte mit Softwareentwicklung. Allerdings fällt auf, dass die Aufteilung bei Gruppe-B deutlich extremer ist als bei Gruppe-A

Während bei Gruppe-A 75% der Teilnehmer regelmäßig mit Softwareentwicklung zu tun haben, haben dies bei Gruppe-B nur $41,6\%$, dafür dann aber schon länger als drei Jahre. Noch deutlicher wird diese Kluft bei dem Vergleich der Personen, die keine Erfahrung oder nur gelegentlich mit Softwareentwicklung zu tun haben. Bei Gruppe-A fallen in diese Kategorie nur 25% und bei Gruppe-B sind es $58,3\%$. Dies spiegelt sich vorrangig in zwei Dreiergruppen wider, die die auch schon zuvor erwähnt wurden. Hier sind es die einzigen beiden Gruppen, die nur aus Teilnehmern mit keiner oder gelegentlicher Erfahrung bestehen.

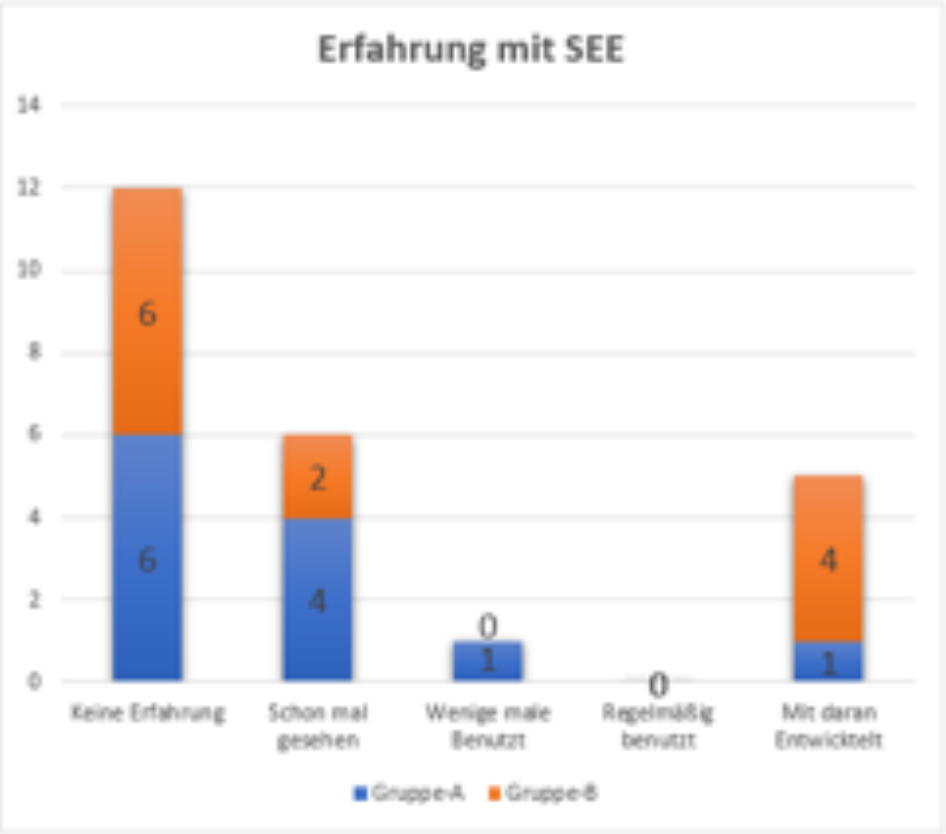

Abbildung 26: Erfahrungen der Probanden mit SEE.

Erfahrung mit SEE Bei der Erfahrung mit SEE gibt es einen deutlichen Trend dazu, dass kaum Erfahrung mit SEE besteht. 50% der Teilnehmer haben keinerlei Erfahrung mit SEE. Das zweitgrößte Segment sind die Teilnehmer, welche SEE lediglich schon einmal gesehen, aber noch nie benutzt haben. Dieses Segment macht 25% aus. Bei dem Vergleich von Gruppe-A und Gruppe-B fällt lediglich auf, dass deutlich mehr Teilnehmer aus Gruppe-B an SEE mit entwickelt haben $(33,3\%)$, bei Gruppe-A waren es nur $8,3\%$. Dafür ist Gruppe-A in dem Segment „Schon mal gesehen“ doppelt so stark vertreten wie Gruppe-B $(33,3\%$ zu $16,6\%)$. Bei einem Blick auf die Rohdaten fällt hierbei auf, dass die beiden Dreiergruppen aus Gruppe-B mit dem SEE-Editor Platz zwei und drei in Bezug auf die benötigte Zeit belegt haben. Die Gruppe auf Platz eins stammt aus Gruppe-A und hat zwei Personen ohne Erfahrung mit SEE und eine, die es schon mal gesehen hat.

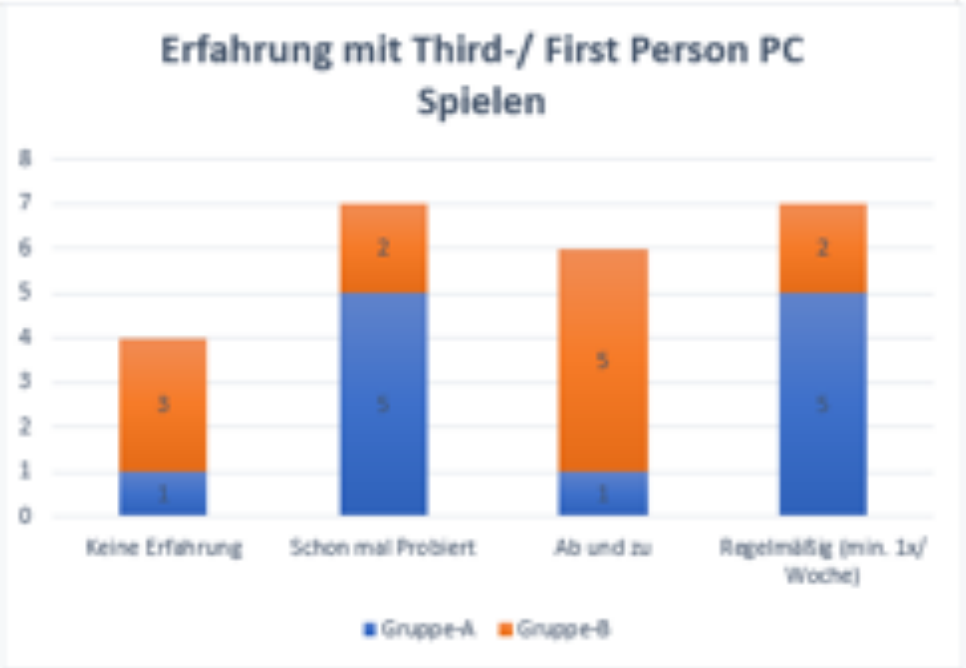

Abbildung 27: Erfahrung mit First-/ Third-Person PC-Spielen.

Erfahrung mit First-/ Third-Person PC-Spielen Bei einem Blick auf die Grafik fällt auf, dass $83,3\%$ der Teilnehmer schon mindestens einmal ein PC-Spiel in First- oder Third-Person gespielt haben. Jedoch ist der Anteil, der Teilnehmer ohne Erfahrung mit PC Spielen, bei Gruppe-B deutlich höher als bei Gruppe-A (25% zu $8,3\%$).

Fazit zur Demografie In Bezug auf die demografischen Auswertungen sind zwar in einigen Aspekten und Besonderheiten aufgefallen, jedoch sind diese nur in geringen Anzahlen zu beobachten. Deshalb lassen sich hieraus noch keine Rückschlüsse auf Einschränkungen der Validität der Daten ziehen.

6.7.2 Zeit und Korrektheit

In diesem Abschnitt wird nun, nachdem die Demografie ausgewertet wurde, zunächst die Zeit, die zum Lösen der Aufgabe benötigt wurde, ausgewertet und danach die Korrektheit der Lösungen ermittelt.

Zeit Als erste Hypothese wurde vermutet, dass die Aufgabe mit dem SEE-Editor schneller zu lösen ist als mit dem Windows-Editor.
Die folgende Grafik soll zunächst einen Überblick über die benötigten Zeiten zur Lösung der Aufgaben geben.

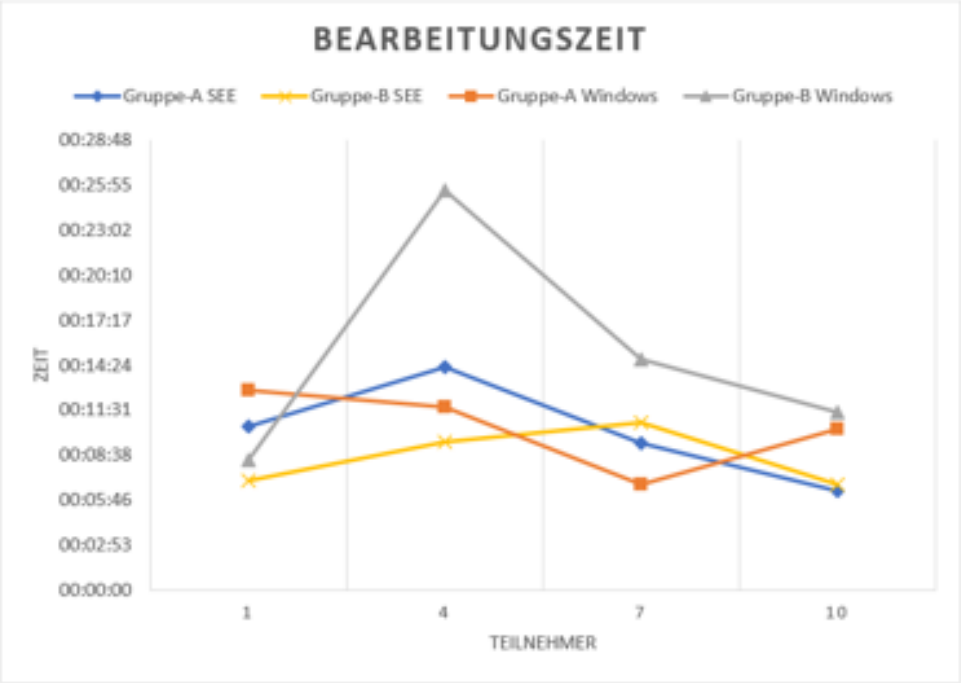

Abbildung 28: Die Bearbeitungszeiten der einzelnen Gruppen.

Auf der Grafik stellen die einzelnen Datenreihen jeweils die Zeiten der einzelnen Dreiergruppen dar. Die Gruppen-A und B werden jeweils in den Windows-Editor und den SEE-Editor aufgeteilt und zunächst getrennt voneinander betrachtet.
Es deutet sich an, dass es einerseits einen generellen Unterschied zwischen Gruppe-A und Gruppe-B gab, andererseits allerdings auch einen Unterschied zwischen dem SEE- und dem Windows-Editor. Betrachtet man nur den Durchschnitt der Daten, war Gruppe-B mit dem SEE-Editor mit durchschnittlich achteinhalb Minuten Bearbeitungszeit am schnellsten. Am zweitschnellsten war Gruppe-A mit dem SEE-Editor in zehn Minuten und sieben Sekunden. Dahinter folgt dann Gruppe-A mit dem Windows-Editor in zehn Minuten und 23 Sekunden und zu guter Letzt folgt noch Gruppe-B mit dem Windows-Editor in fünfzehn Minuten und zwei Sekunden. Bei der letzten Datenreihe muss allerdings beachtet werden, dass es einen Ausreißer gab. Da die zweite Dreiergruppe aus Gruppe-B sich die Einträge im Windows-Editor statt diese mithilfe des Chats zu synchronisieren, sich diese mithilfe des deutschen Buchstabieralphabets durch den Sprachchat abgeglichen hat.
Bildet man den Durchschnitt über alle Gruppen, um den SEE-Editor mit dem Windows-Editor zu vergleichen, so erhält man neun Minuten und achtzehn Sekunden für den SEE-Editor und zwölf Minuten und 43 Sekunden für den

Windows-Editor.
Dies deutet schon darauf hin, dass die Durchführung der Aufgabe mit dem SEE-Editor schneller möglich war als mit dem Windows-Editor.
Nach diesen ersten Betrachtungen wurden die Daten nun noch in einem Boxplot dargestellt, um weitere statistische Schlüsse ziehen zu können.

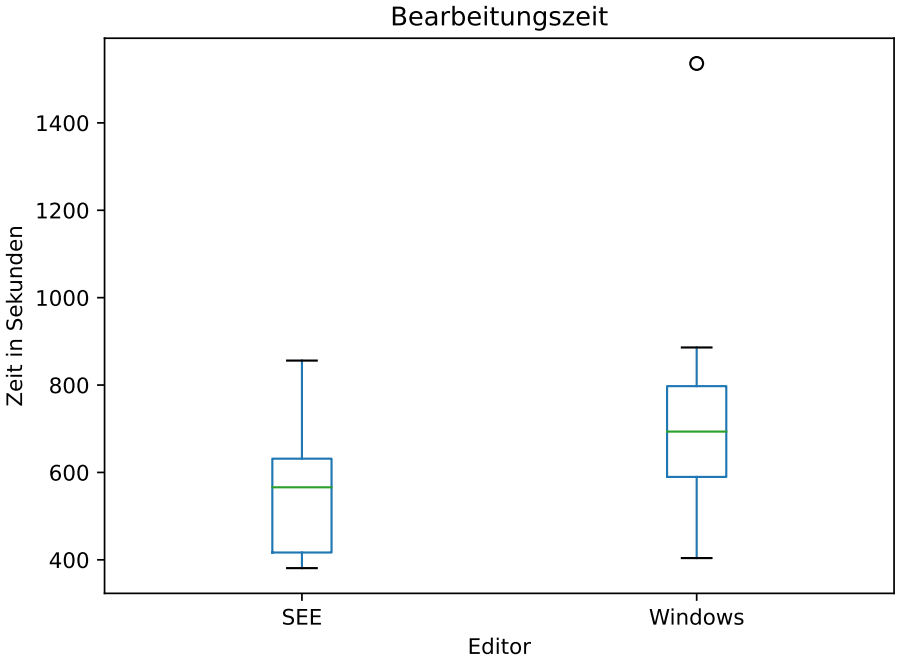

Abbildung 29: Auswertung der Zeiten als Boxplot.

Aus dem Boxplot geht hervor, dass die Bearbeitungszeit mit dem SEE-Editor kürzer ist. Damit wurde die Alternativhypothese zur Bearbeitungszeit bestätigt und die Nullhypothese kann verworfen werden. Mithilfe des *Mann-Whitney-U Test* wurde im Anschluss geprüft, ob der Unterschied statistisch signifikant ist. Um den Mann-Whitney-U Test durchzuführen, wurde die Implementierung von *Scipy* benutzt.[12] Es wird ein p-Wert von $\alpha = 0,05$ gewählt, da dies der gängige Standard in der Softwaretechnik ist. [Sta] Bei der Berechnung wurde ein p-Wert von $\alpha = 0,08$ errechnet. Da dieser Wert größer ist als das gewählte Signifikanzniveau, lässt sich kein statistisch signifikanter Unterschied feststellen. Das bedeutet, dass die Alternativhypothese nicht verallgemeinert werden

Mann-Whitney-U Test: Bestimmt, ob es zwischen zwei Gruppen einen statistisch signifikanten Unterschied gibt, ohne dass die Daten normalverteilt sein müssen.

Scipy: Ein Framework für die Programmiersprache Python, mit dessen Hilfe u. a. statistische Berechnungen durchgeführt werden können.

[12] Scipy Dokumentation zum Mann-Whitney-U Test
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html

kann. Es fällt allerdings auf, dass er nah am gewählten Signifikanzniveau liegt. Da der Boxplot relativ deutlich einen Unterschied zeigt, könnte es sein, dass das Ergebnis an der geringen Anzahl von Daten von $n = 16$ liegt und bei einer umfangreicheren Studie, mit mehr Datensätzen, durchaus ein solcher festgestellt werden könnte.

Korrektheit Als Zweites wurde die Korrektheit der Lösungen ausgewertet. Auch hier wird wieder nur die Dreiergruppen Ebene betrachtet. Die hier untersuchte These lautet: „Die Aufgaben, die mit dem SEE-Editor bearbeitet wurden, wurden korrekter gelöst als die, die mit dem Windows-Editor bearbeitet wurden.

Zunächst musste eine Definition für die Korrektheit gefunden werden. Die Korrektheit der Lösung wurde nicht absolut betrachtet, da es nicht zielführend gewesen wäre, wenn eine Lösung schon bei einem einzigen Fehler als falsch betrachtet würde.
Deswegen wurden die Lösungen in einzelnen Punkten betrachtet. Bewertet wurden diese dann nach folgenden Kriterien:

- Korrektheit eines einzelnen Eintrags.
- Sortierung eines Abschnitts.

Als Eintrag gilt hierbei $Knotenname,\ LOC,\ Kindknotenname$.
Ein Abschnitt wiederum besteht aus drei Einträgen eines Nutzers. Zusätzlich gibt es noch den Abschnitt *Top neun* — hiermit ist die oben in Unterabschnitt 6.3 erwähnte Auflistung der nach LOC sortierten Einträge gemeint.
Ein Eintrag gilt als korrekt, wenn folgende Bedingungen erfüllt sind:

- Der Abschnitt, in dem der Eintrag steht, wurde von dem Teilnehmer selbst erstellt, oder der Eintrag ist bei mindestens zwei Teilnehmern gleich.
- Es gibt keine Tippfehler wie vergessene, hinzugefügte oder vertauschte Buchstaben und die Groß- und Kleinschreibung wurde beachtet.
- Im Namen selbst gibt es keine unnötigen Leerzeichen: $Knoten\ Name \neq KnotenName$
- Die LOC eines Eintrags stimmt mit mindestens einem Eintrag eines anderen Nutzers überein oder wurde von ihm selbst erstellt.

- Der Eintrag ist vollständig.

Werden diese Punkte erfüllt, gibt es pro korrekten Eintrag einen Punkt.
Prinzipiell gilt dabei, dass jeder Teilnehmer nur einen einzigen Abschnitt selbst erstellen kann.
Da die *Top neun* in Kollaboration der Probanden entstand, gilt dieser Abschnitt als Ausnahme. Bei diesen muss bei mindestens zwei Teilnehmern ein identischer Eintrag vorhanden sein, um die Korrektheit zu erfüllen.
Nicht bewertet wurden folgende Punkte:

- Leerzeichen und Zeilenumbrüche, die nicht innerhalb eines Knotennamens oder Kindknotennamens sind. Es konnten also Leerzeichen oder Zeilenumbrüche beliebig hinzugefügt oder weggelassen werden, solange sie nicht ein Wort unterbrechen oder zwei Worte verbinden. Ausgenommen davon waren die LOC und die Zeilennummern, da diese ein eindeutiger Trennpunkt sind.

- Wörter, die an beliebiger Stelle in der Datei hinzugefügt oder gelöscht wurden, sofern sie nicht einem Eintrag zugehörig sind.

- Ob hinter die LOC noch das Wort $Zeilen$ geschrieben wurde oder nicht.

- Ob der Eintrag kommasepariert war oder nicht.

- Ob im Eintrag XML-Tags waren.

- Ob für den Eintrag die richtigen Informationen aus der SEE-City gesucht wurden.

Wenn ein Abschnitt vollständig korrekt ist, konnte pro Abschnitt je ein Zusatzpunkt erreicht werden, wenn die Einträge im Abschnitt die gleiche Reihenfolge haben. Auch hier gilt: wenn ein Teilnehmer einen Abschnitt erstellt hat, gilt er als sortiert. Überdies wurde bei der *Top neun* nicht überprüft, ob die Einträge wirklich absteigend sortiert waren, sondern lediglich, ob sie bei den anderen Teilnehmern gleich sortiert waren. Dies wurde gemacht, da nicht bewertet werden sollte, ob die Nutzer in der Lage sind, Informationen aus der SEE-City korrekt zu extrahieren, sondern um zu bewerten, mit welchem Editor die kollaborative Arbeit besser funktioniert.
Insgesamt gab es so pro Dreiergruppe 66 Punkte zu erreichen. Die detaillier-

te Bewertung der einzelnen Gruppen ist im Anhang im Ordner *Korrektheit* zu finden.
Zur Kontrolle der manuellen Auswertung wurde der Datensatz mit den hier beschrieben Kriterien noch an eine andere Person geschickt, damit diese validieren kann, ob die Auswertung korrekt ist. In der nun folgenden Grafik ist ein Vergleich der Korrektheit des SEE-Editors mit der des Windows-Editors zu sehen.

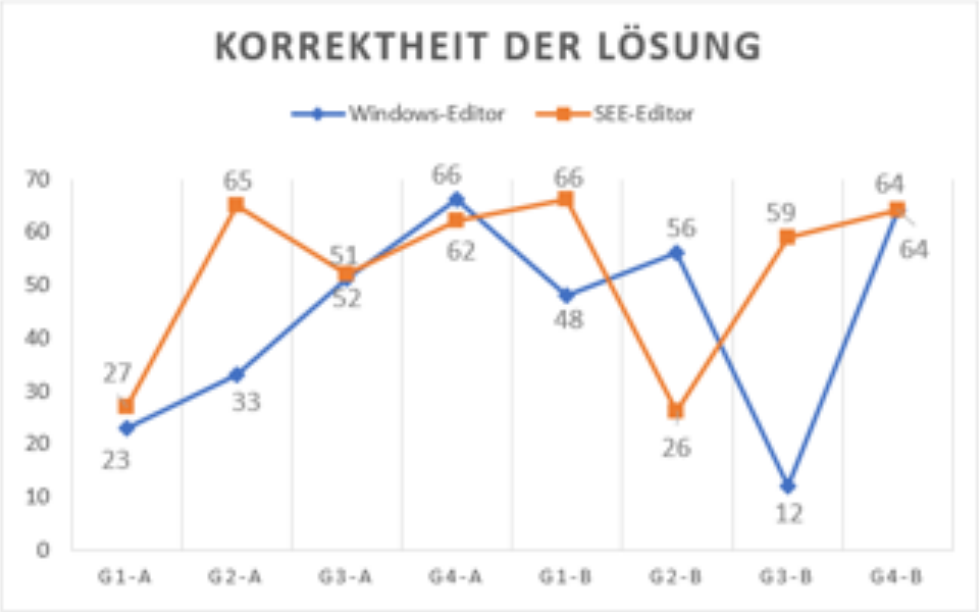

Abbildung 30: Vergleich der Korrektheit der Lösungen pro Gruppe.

Wie in der Grafik zu erkennen, war die Korrektheit im SEE-Editor überwiegend höher zu $62,5\%$. Es gab jedoch auch eine Gruppe, wo der SEE-Editor deutlich schlechter abgeschnitten hat. Im Durchschnitt ist der SEE-Editor mit $52,63$ Punkten um einiges korrekter als der Windows-Editor mit $44,13$ Punkten — dies entspricht einer durchschnittlich höheren Korrektheit von $16,15\%$.

Zwischen Gruppe-A und Gruppe-B gibt es auch einen deutlichen Unterschied, so erreichte A im Durchschnitt $43,25$ Punkte mit dem Windows-Editor und $51,5$ Punkte mit dem SEE-Editor. Gruppe-B erreichte hingegen 45 Punkte mit dem Windows-Editor und $53,75$ Punkte mit dem SEE-Editor.
Interessant ist, obwohl durch den SEE-Editor die Informationen eigentlich synchronisiert und sie damit gleich genauer gesagt korrekt sein sollten, trotzdem in $87,5\%$ der Fälle keine volle Punktzahl erreicht werden konnte. In der weiteren Auswertung ist aufgefallen, dass dies nicht unbedingt an den Teilnehmern

selbst lag, sondern meist an technischen Problemen.
Beim Windows-Editor hingegen sind die Fehler klar auf die Teilnehmer selbst zurückzuführen. Hier gab es zum einen Gruppen, die die Informationen nicht über den Chat durch Kopieren oder Einfügen getauscht haben, sondern durch Abtippen oder Diktieren. Diese haben hauptsächlich kleine Fehler wie Groß- und Kleinschreibung oder akustische Verständnisfehler durch falsche Namen. Es gab allerdings auch Teilnehmer, sowohl in Gruppe-A als auch in Gruppe-B, die vergessen haben, die Informationen der anderen bei sich einzutragen. Dies hat teilweise zu massivem Punktabzug geführt, vorwiegend dann, wenn in der *Top neun* nur Abkürzungen verwendet wurden, aber an den durch die Abkürzungen verwiesenen Stellen keine Einträge vorhanden waren.

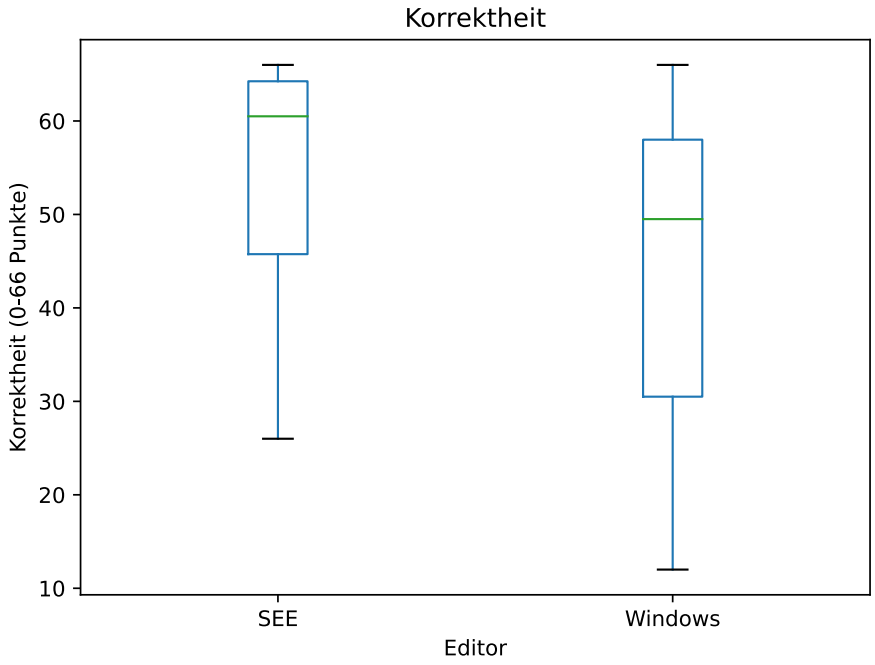

Abbildung 31: Korrektheit der Lösungen als Boxplot.

Wie im Boxplot zu erkennen, ist die Korrektheit im Median im SEE-Editor besser als die im Windows-Editor. Die statistische Signifikanz allerdings noch geringer als bei der zeitlichen Auswertung. Der p-Wert ist hier $\alpha = 0,16$. Ob hier eine größere Testgruppe ein signifikanteres Ergebnis liefern würde, ist unklar. Die Alternativhypothese konnte zwar für dieses Experiment gezeigt, jedoch nicht verallgemeinert werden.

Fazit zur Korrektheit und Zeit Nach der Auswertung ist klar, dass die Lösung der Aufgabe mithilfe des SEE-Editors sowohl zeitlich schneller als auch korrekter war. Dieses Ergebnis ist aber nicht statistisch signifikant. Somit wurden zwar beide Alternativhypothesen für dieses Experiment bestätigt, konnten aber aufgrund der fehlenden Signifikanz nicht verallgemeinert werden.

6.7.3 Benutzbarkeit

Um These drei zu evaluieren, muss die Benutzbarkeit ausgewertet werden. Da die Benutzbarkeit ein subjektives Maß ist, wurde sie versucht, einheitlich über den SUS zu erfassen. Die aufgestellte Hypothese war, dass der SEE-Editor für die gestellte Aufgabe eine bessere Benutzbarkeit ausweist.
Im folgenden Abschnitt werden zunächst einmal die erfassten Daten vorgestellt und im Anschluss der SUS ausgewertet, um die Benutzbarkeit zu erfassen.

Die Daten Da durch die insgesamt 20 Fragen pro Teilnehmer bei den 24 Teilnehmern insgesamt 480 Datenpunkte entstehen, werden diese nicht als Tabelle aufgelistet, da dies unübersichtlich wäre.
Stattdessen werden sie für eine einfachere Übersicht in vier Boxplots abgebildet. Die kompletten Daten sind im Anhang unter *Auswertung.xlsx* zu finden.

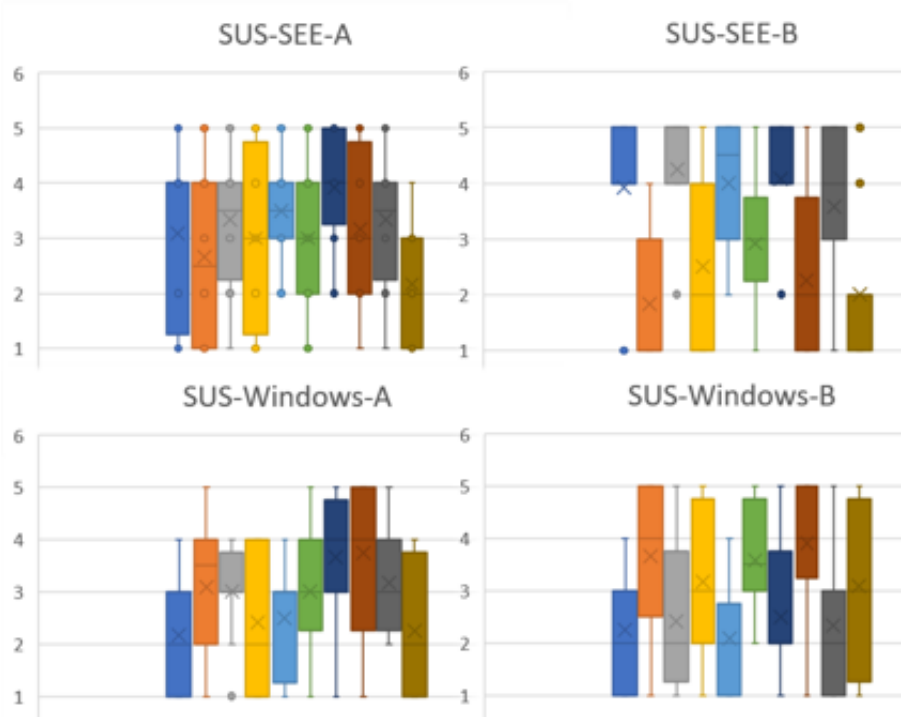

Abbildung 32: Rohdaten zu den SUS Fragen. Jede Box stellt eine Frage dar.

Es ist zu erkennen, dass es einen Unterschied zwischen Gruppe-A und B gibt. Während bei dem SUS von Gruppe-A zu SEE teilweise deutlich unterschiedliche Antworten geben wurden, ist es bei Gruppe-B eindeutiger. Dass abwechselnd viele und wenig Punkte vergeben wurden, liegt an der Struktur des SUS, denn jede zweite Frage ist eine negative Frage, also eine, bei der weniger Punkte vergeben werden, wenn das System gut ist.
Bei dem SUS zu Windows wird es dann auch bei Gruppe-A deutlicher. Es werden bei beiden Gruppen bei den negativen Fragen häufig mehr Punkte vergeben als bei den positiven.

Die Auswertung Zur Auswertung werden die Rohdaten mit folgender Formel auf einer Skala von 0 bis 100 abgebildet.

$$Score = 2,5 * (20 + \sum_{i=1}^{10} (-1)^{i+1} * S_i)$$

Nach dem Anwenden dieser Formel erhalten wir folgende SUS Werte:

	Windows-Editor	SEE-Editor
T1	50	70
T2	45	60
T3	42,5	50
T4	22,5	17,5
T5	45	50
T6	77,5	75
T7	70	57,5
T8	62,5	82,5
T9	77,5	20
T10	25	90
T11	45	47,5
T12	37,5	75
T13	72,5	70
T14	47,5	40
T15	30	42,5
T16	17,5	92,5
T17	25	100
T18	25	100
T19	17,5	77,5
T20	12,5	92,5
T21	5	67,5
T22	70	62,5
T23	55	27,5
T24	47,5	77,5

Abbildung 33: Die Auswertung des SUS als Tabelle.

Es gilt, eine gute Benutzbarkeit liegt vor, wenn der SUS Wert über 70 liegt. Ist der Wert unterhalb von 50 gibt es starke Probleme mit der Nutzbarkeit.[Ryt14] Wie in der Tabelle zu erkennen, gibt es in deutlich abweichende Meinungen zur Benutzbarkeit der Systeme. Im Median liegt der SUS bei dem SEE-Editor bei 68, 75 und bei dem Windows-Editor bei 45. An dieser Stelle sei angemerkt: Die Benutzbarkeit bezieht sich nicht nur auf den Editor selbst, sondern auf das komplette System zur Lösung der Aufgabe verwendet wurde. Also bei dem Windows-Editor wurde mit bewertet, das die Informationen aus SEE abgeschrieben und dann irgendwie manuell synchronisiert werden mussten. Bei dem SEE-Editor ist neben dem Editor selbst nur der Aspekt mit bedacht, dass die Informationen aus der Stadt gelesen werden mussten.
Somit ist der SEE-Editor knapp unterhalb der 70 Punkte, bei denen ein System noch als benutzbar gilt. Dies war auch zu erwarten, da der Editor noch ein paar Probleme mit der Benutzbarkeit hat, wie etwa die defekte Scrollfunktion. Beim Windows-Editor gibt es mit einem Wert von unter 50 noch größere Probleme mit der Benutzbarkeit[Ber16]. Auch dies war zu erwarten, da es relativ umständlich ist, wenn man die Ergebnisse der anderen erst erfragen muss, bevor man damit weiter arbeiten kann.
Abschließend ist der SUS Wert nun noch einmal als vergleichender Boxplot abgebildet.

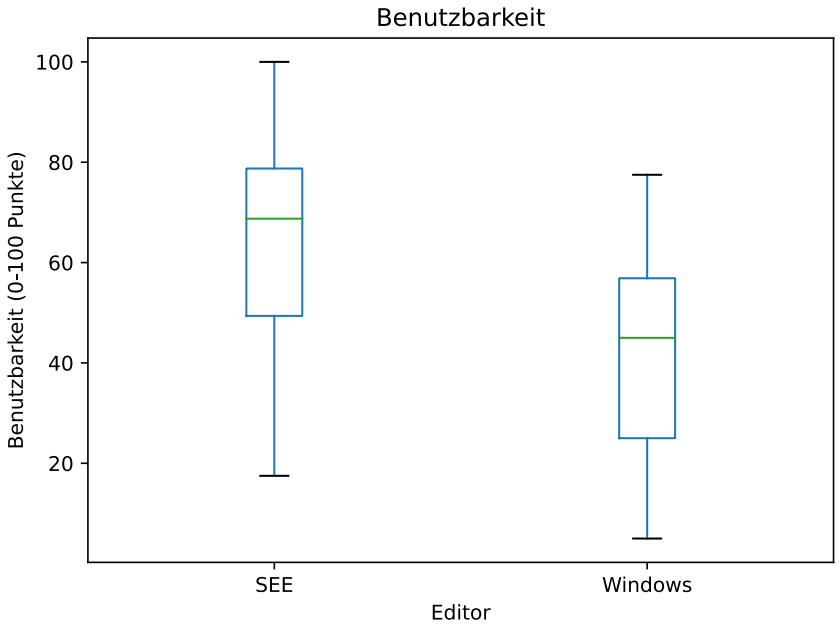

Abbildung 34: Auswertung der Benutzbarkeit.

Dieser Plot bestätigt noch das oben genannte Ergebnis, dass der SEE-Editor bei dem SUS Wert im Median höher ist als der Windows-Editor. Nach einer Auswertung mit dem Mann-Whitney-U Test wurde bestätigt, dass das Ergebnis statistisch signifikant ist, da der p-Wert bei $\alpha = 0,0016$ liegt, was deutlich unterhalb der Signifikanzschwelle liegt. Deswegen kann die Nullhypothese an dieser Stelle sowohl als bestätigt, wie auch als allgemeingültig erkannt werden.

6.7.4 Fragen zur Aufgabe und weitere Anmerkungen

Zuletzt sollen noch die drei Fragen zur Aufgabe ausgewertet werden. Diese Fragen gehören zu keinem validierten Fragebogen. Danach werden noch die Anmerkungen der Nutzer ausgewertet.

Fragen zur Aufgabe Zuletzt wurden noch drei Fragen zum empfundenen Aufwand, der mit der Lösung der Aufgabe verbunden war. Während Frage eins und zwei relativ unabhängig voneinander sein sollten, hängt Frage drei mehr mit den ersten beiden zusammen, ergänzt diese jedoch noch um eventuell zu vor nicht erfasste Aspekte. Die erste Frage war:„Ich finde es einfach, mit diesem System mit mehreren Leuten Informationen zusammenzutragen.“.

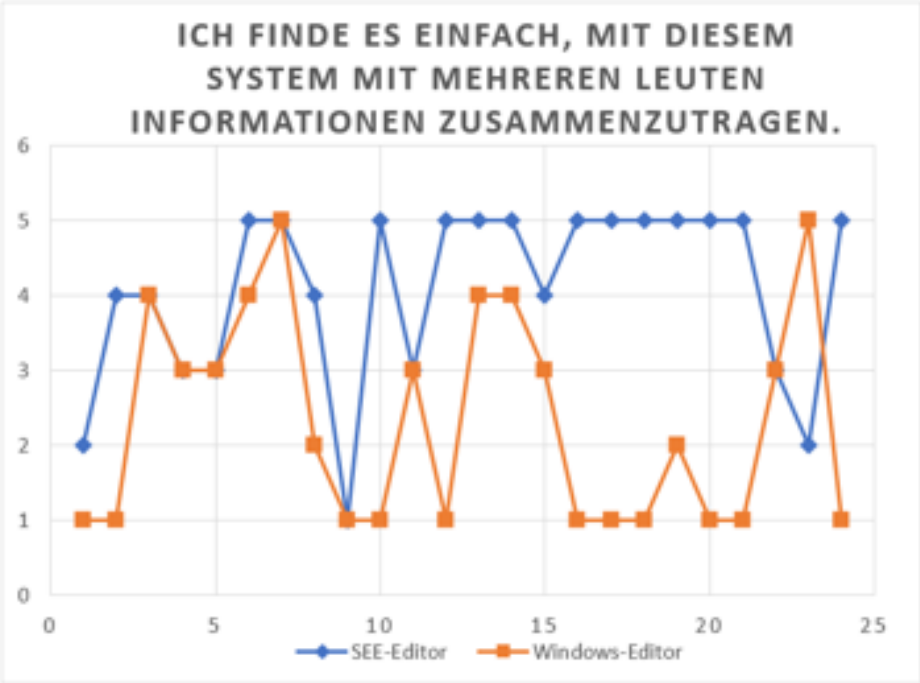

Abbildung 35: Antworten auf Frage eins, je mehr Punkte desto besser.

Bei einem Blick auf die Daten fällt auf, dass nur eine Person es einfacher findet, die Daten mit dem Windows-Editor zu synchronisieren. Alle anderen finden es einfacher oder gleich schwer mit dem SEE-Editor.

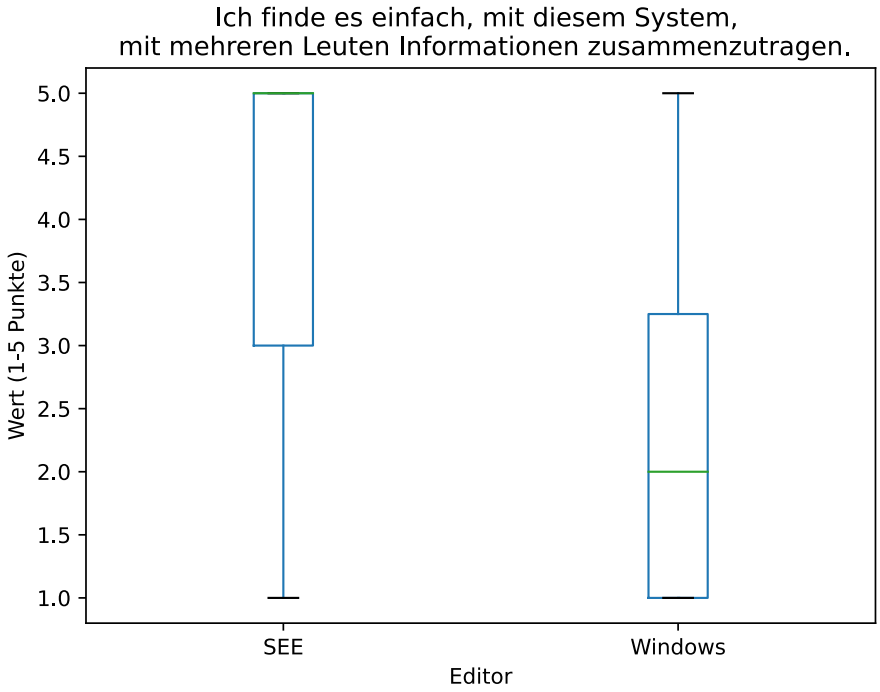

Abbildung 36: Boxplot zur ersten aufgabenspezifischen Frage.

Der Median bestätigt die zuvor angeführte Vermutung. Der SEE-Editor Median ist deutlich höher als der von dem Windows-Editor. Der Mann-Whitney-U Test ergibt eine sehr starke statistische Signifikanz mit einem p-Wert von $\alpha = 0,0000554$. Dies war auch die Vermutung, da es für einen Nutzer einen größeren Aufwand bedeutet, wenn er sich die Ergebnisse der anderen Nutzer aktiv erfragen muss, als wenn sie automatisch synchronisiert werden.

Als Nächstes wird die Frage „Ich finde es einfach, mit diesem System mit mehreren Leuten Informationen zu sortieren. “ausgewertet.

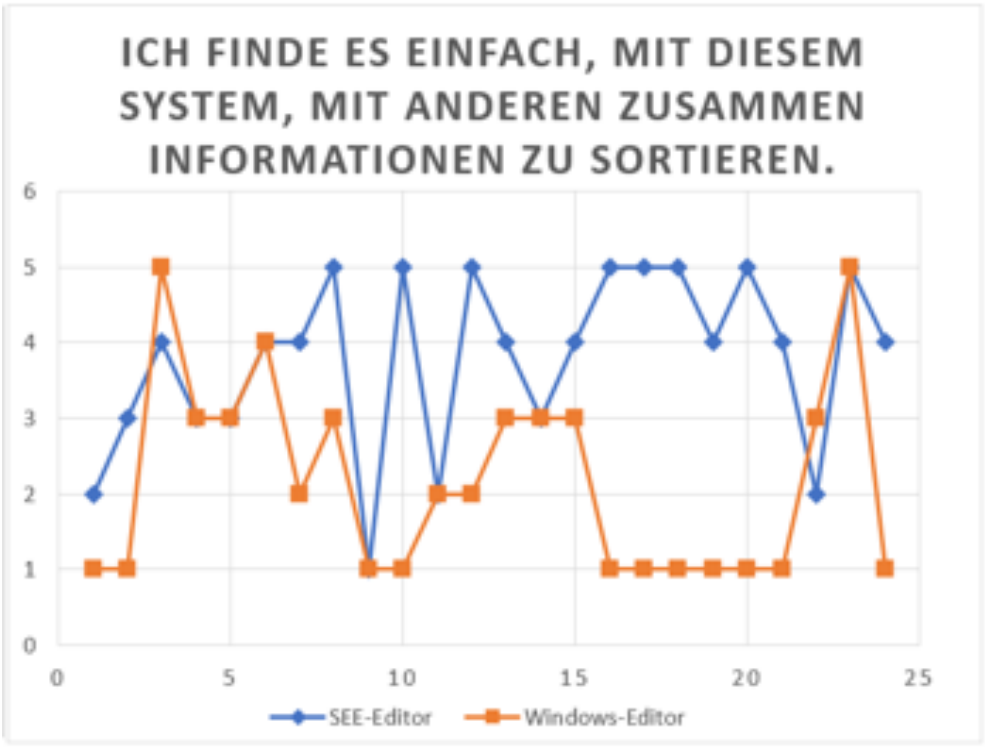

Abbildung 37: Antworten auf Frage zwei, je mehr Punkte, desto besser.

Interessant ist im Vergleich zu Frage eins gibt es deutlich weniger fünf Punkte Bewertungen für den SEE-Editor. Dies liegt vermutlich daran, dass die Probleme in Bezug zur Benutzbarkeit bei dem Sortieren deutlich stärker auffallen. Dies liegt vermutlich daran, dass bei dem Sortiervorgang häufig im Code-Window hoch und runtergescrollt werden muss. Da dies noch mit Problemen belastet ist, ist es nicht so angenehm wie das Einfügen von neuen Daten, bei dem vorrangig ohne Scrollen gearbeitet werden konnte.
Dieser Wandel ist noch eindeutiger in dem Boxplot zuerkennen.

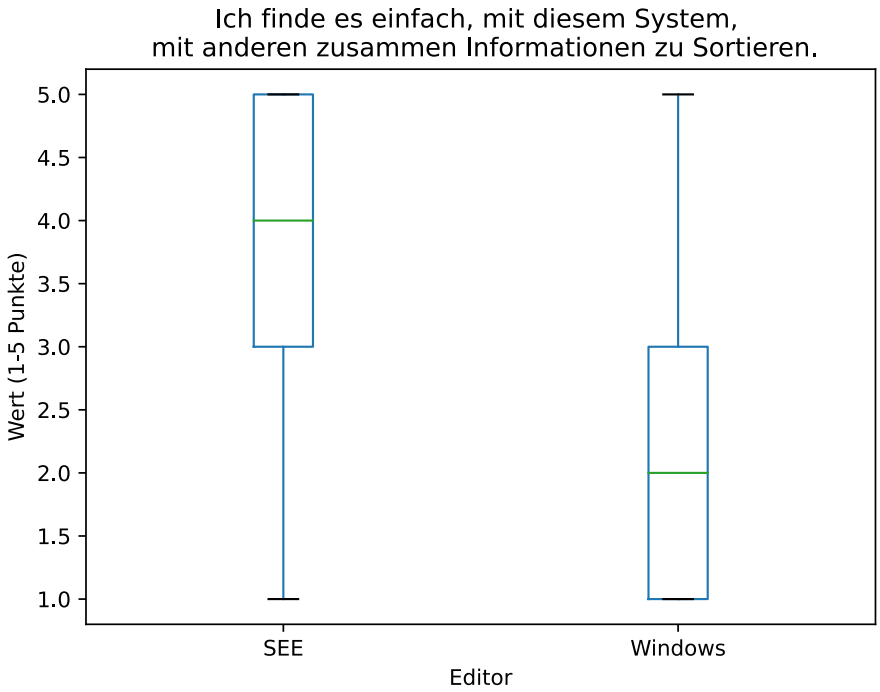

Abbildung 38: Antworten auf Frage zwei, je mehr Punkte, desto besser.

Im Boxplot ist deutlich zu erkennen, dass sich zwar der Windows-Editor nicht wirklich verändert hat, jedoch der SEE-Editor deutlich an Leistung eingebüßt hat. Wie bereits oben beschrieben, könnte die an der deutlich intensiveren Nutzung des Editors bei dem Sortiervorgang und damit einhergehende Probleme mit der Benutzbarkeit zurückzuführen sein. Interessant ist, dass der Windows-Editor kaum Veränderungen im Vergleich zu Frage eins zeigt. Dies könnte vermutlich darauf zurückzuführen sein, dass die Teilnehmer nicht so gut die Daten kollaborativ sortieren können, da die Ergebnisse der anderen nicht angezeigt werden. Auch, wenn ein einzelner Sortiervorgang an sich in dem Windows-Editor vermutlich besser funktioniert hat.
Eine Überprüfung der Ergebnisse mittels des Mann-Whitney-U Test hat allerdings auch hier wieder eine sehr starke statistische Signifikanz der Ergebnisse gezeigt, mit einem p-Wert von $\alpha = 0,00007$.

Als letzte Frage wurde folgende gestellt:„Ich finde, es war einfach, die gegebene Aufgabe mit dem System zu lösen. “Bei dieser Frage, als Ergebnis etwas in der Mitte der ersten beiden Fragen erwartet, da beide Fragen hier einfließen, jedoch auch noch von anderen Aspekten beeinflusst werden könnten.

Abbildung 39: Antworten auf Frage drei, je mehr Punkte, desto besser.

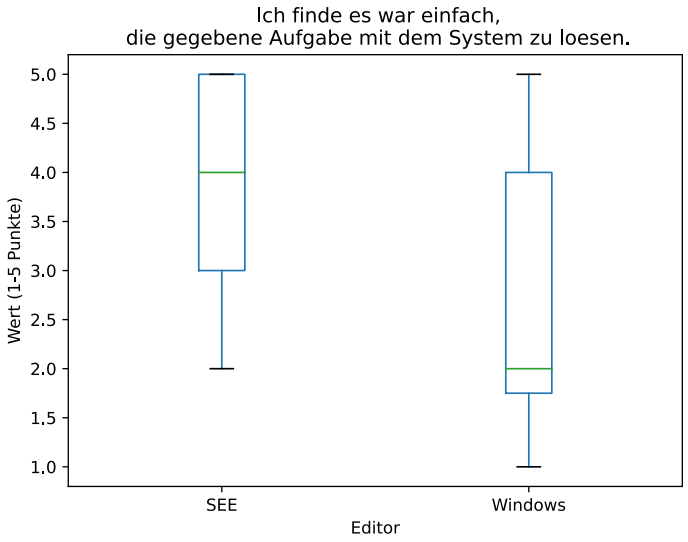

Abbildung 40: Antworten auf Frage drei, je mehr Punkte, desto besser.

Es fällt auf, dass der SEE-Editor im Boxplot fast gleich geblieben ist, lediglich die Ausreißer sind weniger extrem geworden. Bei dem Windows-Editor fällt auf, dass dieser ein wenig besser bewertet worden ist, allerdings liegt der Median noch bei zwei, wie ebenfalls bei Frage eins und zwei.

Die statistische Signifikanz der Auswertung ist etwas schlechter geworden, liegt jedoch mit einem p-Wert von $\alpha = 0,0012$ immer noch deutlich unter $\alpha = 0,05$.

Fazit zu den eigenen Fragen Abschließend wurde herausgefunden, dass bei allen drei Fragen der SEE-Editor besser abgeschnitten hat und dies auch durch statistische Signifikanz verallgemeinert werden kann.

Anmerkungen der Nutzer Insgesamt hatten 10 Nutzer sinnvolle Kommentare abgegeben. Die meisten Teilnehmer finden die Idee gut, allerdings merken einige an, dass es noch Probleme mit der Benutzbarkeit in bestimmten Belangen, wie etwa mit dem Scrollen gibt. Ein paar haben auch angemerkt, dass es Synchronisationsprobleme bei ihnen mit dem SEE-Editor gab. Zum Windows-Editor wurde angemerkt, dass die Synchronisierung aufwendig ist.

6.8 Bedrohungen für die Validität der Evaluation

In diesem Abschnitt werden verschiedene von Ohlund und Chong-ho [Bar] sowie von Bhandari [Bha20] genannte Faktoren überprüft, die die Validität der Evaluationsergebnisse bedrohen könnten. Zunächst werden Faktoren für die interne Validität und im Anschluss Faktoren für die externe Validität beleuchtet.

6.8.1 Interne Faktoren

Die internen Faktoren, die eine Bedrohung für die Validität der Studienergebnisse darstellen, sind Faktoren, welche durch das Experiment Design direkt beeinflusst werden können. Von Ohlund und Chong-ho [Bar] wurden insgesamt neun Faktoren für die Bedrohung der internen Validität genannt.

Historie/ Reifung Mit diesen Faktoren sind zum einen Ereignisse gemeint, die zwischen den beiden Aufgaben oder auch zwischen verschiedenen Gruppen passiert sind. Zum anderen werden damit auch die Lerneffekte der Teilnehmer gemeint.
In dieser Studie muss der erste Punkt in zwei Perspektiven betrachtet werden, zum einen Ereignisse, die bei einer Gruppe zwischen Aufgabe eins und zwei stattgefunden haben und Ereignisse, die zwischen den einzelnen Gruppen stattgefunden haben.

Da bis auf eine Gruppe alle Gruppen beide Aufgaben ohne große Pause dazwischen gelöst haben, kann vermutlich ein beeinflussendes Ereignis zwischen den beiden ausgeschlossen werden. Hingegen zwischen den einzelnen Gruppen ist mehr Zeit verstrichen und es gab an zwei Stellen Änderungen.

Zum einen wurde nach der ersten Aufgabe der zweiten Gruppe eine Unterbrechung eingelegt, da es Probleme mit der Netzwerkgeschwindigkeit gab und eine dementsprechende Änderung in der Software zwischen den beiden Durchläufen durchgeführt werden musste.
Zum anderen wurde nach dem Durchlauf von Gruppe vier noch eine Änderung vorgenommen, welche die mit der ersten Änderung einhergehenden Fehler in SEE behoben haben. Diese Änderungen teilen sich in zwei Aspekte. Zum Ersten wurde der Fehler behoben, dass die Stadt die Teilnehmer unter den Boden drückt, diese Änderung hat Aufgabe eins wie zwei gleichermaßen beeinflusst.
Zum Zweiten wurden die Fehler behoben, die dazu geführt haben, dass der Chat während des Tippens geöffnet wird oder sich der Spieler beim Tippen bewegt. Diese Änderungen haben nur Einfluss auf den SEE-Editor.
In den Daten zur Korrektheit ist dabei kein auffälliger Einfluss zuerkennen. Bei der zeitlichen Auswertung hingegen könnte es einen Einfluss bei Gruppe-A auf die Zeiten des SEE-Editors gegeben hat, da die beiden Gruppen nach der Änderung schneller sind, die beiden vor der Änderung. Bei Gruppe-B hingegen ist jedoch kein eindeutiger Einfluss zu erkennen.

Die **Reifung** der Teilnehmer ist vermutlich ein Einfluss, da die meisten Teilnehmer keine Erfahrung mit SEE hatten. Dieser Faktor wurde versucht, durch das Aufteilen in zwei Gruppen, die jeweils die Editoren in unterschiedlicher Reihenfolge genutzt haben, auszugleichen.
Bei Gruppe-B, die den SEE-Editor als Zweites benutzt haben, gab es jedoch deutlich mehr Teilnehmer, welche mit SEE bereits intensive Erfahrung durch die Entwicklung dieser hatten. Bei diesen könnte der Lerneffekt stärker ausgefallen sein als bei den anderen beiden Gruppen aus B, da sie im ersten Durchlauf lediglich die Aufgabe verstehen mussten, jedoch nicht SEE selbst. Aufgrund der geringen Datenmenge von zwei Gruppen lässt sich dies jedoch nicht sicher sagen.

Tests Es könnte sein, dass die Reihenfolge, in der die Aufgaben ausgeführt wurden, einen Einfluss auf die Ergebnisse nimmt. Um diesen Effekt zu verhindern, wurden die Aufgaben von den beiden Gruppen in unterschiedlicher Reihenfolge ausgeführt.

Instrumentierung Hier werden Änderungen am Instrument, den Beobachtern oder den Bewertungsmaßstäben betrachtet. Am Instrument SEE selbst gab es Änderungen, wie bereits im Abschnitt oben beschrieben. Da der Beobachter immer der gleiche war und lediglich für die Einführung und Helfen

bei Problemen verantwortlich war, hatte dieser vermutlich keinen großen Einfluss auf die Ergebnisse. Um Unterschiede in der Einführung zu vermeiden, wurde diese mithilfe einer kurzen Präsentation gehalten, welche sich über die Probanden hinweg nicht geändert hat. Die Bewertungsmaßstäbe wurden nicht geändert, waren somit für alle Durchführungen gleich.

Statistische Regression Es ist damit gemeint, dass die Auswahl der Probanden auf Grundlage von extremen Eigenschaften einen Einfluss auf die Ergebnisse haben kann. Als Beispiel wurde hier angeführt:„Geben Sie mir vierzig der schlechtesten Schüler und ich garantiere Ihnen, dass sie sich sofort nach meiner Behandlung verbessern werden.“[Bar] Da es bei dieser Studie keine Auswahl der Teilnehmer nach bestimmten Eigenschaften gab, gibt es vermutlich keinen nennenswerten Effekt. Dieser Effekt wurde zusätzlich durch die Einteilung in Dreiergruppen reduziert. Der Datensatz ist allerdings vermutlich zu klein, um einzelne Ausreißer auf den Zusammenhang mit einer demografischen Ausprägung zu bringen, da es diese auch an unterschiedlichen Stellen gab.

Auswahl der Probanden Die Auswahl der Teilnehmer in Gruppe-A und B könnte einen Einfluss auf die Ergebnisse haben. Da die Gruppen nicht bewusst nach Eigenschaften eingeteilt wurden, ist dieser Effekt vermutlich nicht extrem, es gibt jedoch ein paar Unterschiede in der Verteilung der Demografie, wie bereits im Auswertungsabschnitt erwähnt. Da die Gruppen allerdings beide die gleichen Aufgaben erfüllt haben, nur in einer unterschiedlichen Reihenfolge, ist dieser Effekt nur in Bezug auf den Lerneffekt respektive Vorwissen anwendbar. In der Hinsicht, dass dieses zu Vor- oder Nachteilen bei der Lösung der ersten Aufgabe im Vergleich zur zweiten Aufgabe haben könnte. Es lässt sich aber auch an dieser Stelle kein eindeutiger Einfluss auf die Zeit oder Korrektheit feststellen.

Experimentelle Sterblichkeit Dies bezeichnet den Effekt, dass häufig einige Teilnehmer die Studie beginnen, aber nicht beenden. Wenn dies vorwiegend in einer bestimmten Gruppe auftritt, könnte es das Ergebnis beeinflussen. Da alle Teilnehmer, die das Experiment gestartet haben, es auch beendet haben, gibt es hier keinen Einfluss auf das Ergebnis.

Zusammenhang zwischen Auswahl und Reifung der Probanden Es könnte sein, dass, wenn die Probanden in Gruppe-A und B nach bestimmten Eigenschaften sortiert gewählt worden sind, dass dies einen Einfluss auf den

Lerneffekt hat. Dieser Effekt wurde versucht zu verhindern, indem die Teilnehmer nicht nach bestimmten Eigenschaften einer Gruppe zugewiesen wurden. An dieser Stelle kann allerdings wieder auf den im Abschnitt **Historie/ Reifung** beobachteten Fall der Erfahrung mit SEE verwiesen werden, dieser könnte dort einen Einfluss genommen haben.

John-Henry-Effekt Dieser Effekt betitelt das Phänomen, dass Personen in einem Testumfeld bessere Leistung abgeben als im normalen Umfeld. Da in dieser Studie allerdings beide Aufgaben unter Testbedingungen erbracht wurden, gibt es hier keinen Einfluss auf den Vergleich.

6.8.2 Externe Faktoren

Die externen Faktoren, die zu einer Bedrohung der Validität der Studienergebnisse führen können, können nicht unbedingt durch das Experiment Design ausgeschlossen werden. Als externe Faktoren wurden zwei Faktoren von Bhandari [Bha20] betrachtet.

Repräsentanz der Testgruppe Mit diesem Faktor ist gemeint, repräsentieren die Teilnehmer der Studie die Allgemeinheit.
In diesem Fall ist klar, dadurch, dass vor allem sehr junge Menschen an der Studie teilgenommen haben, ist diese Demografie nicht auf die Allgemeinheit zu verallgemeinern. Die ausgewählten Probanden repräsentieren eher ein studentisches Nutzerumfeld. In Bezug zu der Zielbranche von SEE sind die demografischen Hintergründe der Teilnehmer vermutlich nicht repräsentativ.

Einfluss von Umweltfaktoren Unter diesen Faktor fallen alle möglichen Einflüsse, die durch das Umfeld bestimmt werden. An dieser Stelle ist vermutlich der interessanteste Effekt, dass die meisten Teilnehmer den Experiment-Leiter mehr oder weniger persönlich kannten. Dies könnte die Bewertung der verglichenen Systeme positiv oder negativ beeinflusst haben. Dieser Faktor kann allerdings nicht mit Sicherheit überprüft werden, da die Probanden dies eventuell nicht bemerkt haben oder auch teilweise nicht zugeben würden.

6.9 Fazit zur Evaluation

Es fällt auf, dass der SEE-Editor in allen Belangen besser abgeschnitten hat. Jedoch kann eine statistische Signifikanz weder in Bezug auf die zeitliche Auswertung noch bei der Auswertung der Korrektheit nachgewiesen werden. Dies könnte primär in Bezug auf die Zeit an der relativ kleinen Teilnehmerzahl liegen. Bei der Korrektheit hingegen müssten vermutlich noch einige Fehler im Programm korrigiert werden, damit dies nicht unregelmäßige Inkonsistenzen erzeugt.

In Bezug auf die Benutzbarkeit wurde nachgewiesen, dass der SEE-Editor besser abschneidet im Hinblick auf die kollaborative Arbeit. Dieses Ergebnis ist auch statistisch signifikant.
Vermutlich liegt dies daran, dass die Probengröße hier $n = 48$ ist und nicht $n = 16$ wie bei der Auswertung der Zeit. Dies war deswegen unterschiedlich, da bei der Auswertung der Zeit immer nur eine Zeit pro Dreiergruppe ausgewertet wurde, da diese bei einer Gruppe immer gleich ist. Der SUS hingegen konnte für jeden Teilnehmer und jedes System einzeln ausgewertet werden, da jeder Teilnehmer eine eigene Meinung zu dem System hat.
Es wurde jedoch auch herausgefunden, dass der SEE-Editor zum einen noch nicht immer korrekt funktioniert und zum anderen auch noch ein paar Probleme in Bezug auf die Benutzbarkeit hat, hier sei vorwiegend die Scrollfunktion angemerkt.

Kapitel 7 AUSBLICK

7.1 Begrenzungen

Zurzeit unterliegt der Code-Editor einigen Begrenzungen, diese werden im Folgenden vorgestellt. Derzeit gibt es noch Fehler in der Synchronisierung, dadurch ist nicht in jedem Zustand sichergestellt, dass jeder Nutzer den gleichen Text vorliegen hat. Außerdem gibt es einen Fehler, dass der Cursor eines anderen Klienten in bestimmten Situationen ans Ende der Zeile springt, wenn ein Teilnehmer mit der Taste *Enter* eine neue Zeile einfügt.
Zudem gibt es noch ein paar Funktionsbeschränkungen. Es ist aktuell möglich, die Zeilennummern zu bearbeiten und die Zeilennummern werden nach dem Einfügen einer neuen Zeile nicht aktualisiert. Des Weiteren wird das Syntax-Highlighting nicht im Hintergrund aktualisiert, sondern lediglich auf explizite Anfrage des Nutzers. Außerdem ist das Speichern der Dateien nur innerhalb der gleichen Datei möglich, die auch geöffnet wurde. Zuletzt ist es auch nicht möglich, neue Dateien zu erstellen oder leere Dateien zu öffnen.

7.2 Weitere Ideen

In der Zukunft ist geplant, die Fehler in Editor zu beheben und noch ein paar Funktionen zu ergänzen. Es könnte ergänzt werden, dass die Zeilennummern in einem eigenen Textfeld dargestellt werden, sodass diese nicht bearbeitet werden können und einfacher nach dem Einfügen einer neuen Zeile aktualisiert werden. Zudem könnte das Syntax-Highlighting optimiert werden, sodass dieses automatisch im Hintergrund neu berechnet und angezeigt werden kann. Eine weitere sinnvolle Ergänzung wäre ein Hervorheben, welcher Nutzer an welcher Stelle im Text Änderungen vornehmen möchte respektive bereits vorgenommen hat. Dazu könnte zum einen für jeden Nutzer in unterschiedlicher Farbe der Cursor der anderen Nutzer eingeblendet werden und zum anderen die bearbeiteten Stellen in der Farbe des jeweiligen Nutzers hinterlegt werden. Als Idee für die Umsetzung könnte der Nutzer, der eine Stelle zuletzt geändert hat, entweder nachträglich an der *SiteID* im CRDT-Eintrag erkannt werden oder direkt bei einem *ChangeEvent* markiert werden, auch hier kann an der *SiteID* erkannt werden, von wem die Änderung stammt. Zuletzt wäre eine sinnvolle Erweiterung, dass der Nutzer neue Dateien erstellen kann. Diese könnten dann auch direkt in der Software-Stadt eingeblendet werden. Des Weiteren könnte das Speichern bestehender Dateien unter anderem Namen ermöglicht werden, wobei diese ebenfalls dann in die Stadt einfügt

werden sollten. Zudem könnte die Lade- und Reaktionszeit eventuell deutlich verbessert werden, indem bei der Übertragung der Änderungen zwischen den Klienten die *prePosition* weggelassen wird.

Kapitel 8 ABSCHLUSS FAZIT

Abschließend lässt sich feststellen, dass der Aufwand, den es bedeutet, einen mehrbenutzerfähigen Editor innerhalb der SEE-Umgebung einzubauen, deutlich unterschätzt wurde. Interessant ist an dieser Stelle auch, dass die Kernaufgabe, Texte zu synchronisieren, einfacher in der Umsetzung ist, als zunächst vermutet. Dagegen war der deutlich größere Aufwand das Integrieren der GUI und der Schnittstelle zum CRDT. An dieser Stelle sind die meisten komplizierten und aufwendig zu behebenden Fehler entstanden und existieren zum Teil immer noch.

Gleichwohl wurde ein zum Großteil funktionaler Editor erschaffen, der im Vergleich zu einem Standard Windows-Editor besser abgeschnitten hat. Zwar war es im Rahmen dieser Arbeit nicht möglich, allgemeingültig zu zeigen, dass Nutzer mit diesem Editor schneller zu einer Lösung kommen, welche dann auch korrekter ist. Es ließ sich jedoch eine Andeutung auf den Trend hin erkennen, dass zumindest die Schnelligkeit in einer Studie mit mehr Teilnehmern gezeigt werden könnte.
Bei der Korrektheit wurde festgestellt, dass der Editor vermutlich auch deutlich besser abschneiden könnte, wenn die Fehler des Editors behoben worden sind. Denn es ist aufgefallen, dass die meisten Fehler innerhalb des SEE-Editors vermutlich auf eine fehlerhafte Synchronisierung zurückzuführen sind, während die Fehler beim Windows-Editor vermutlich meist durch eine schlechte Kommunikation zwischen den Nutzern entstanden sind.

In Bezug auf die Benutzbarkeit des Systems in Hinblick auf die gemeinsame Datenerhebung konnte allerdings gezeigt werden, dass der SEE-Editor besser bewertet wurde als der Windows-Editor. Es gibt zwar auch an dieser Stelle noch deutliches Optimierungspotenzial, so ist vorrangig die fehlerhafte Synchronisierung bei einigen aufgefallen sowie auch die Tatsache, dass das Scrollen Probleme bereitet, von vielen angemerkt worden.

Kapitel A GLOSSAR

ASCII American Standard Code for Information Interchange. Dies ist eine Tabelle, in der festgehalten wird, wie ein Zeichen in verschiedenen Formaten übertragen wird. 28

Beam Senkrecht nach oben gerichtet Lichtstrahlen zur Markierung einer bestimmten Stelle. 6

Event Eine Unity Klasse, welche unter anderem Peripherie Eingaben erkennt und für andere Programmteile nutzbar machen kann. Zusätzlich können mittels eines Events auch eigene Änderungen zwischen verschiedenen Klassen ausgetauscht werden. 26, 27, 29

First-/ Third-Person PC-Spiele Computerspiele, bei denen der Spieler aus der Perspektive der Spielfigur oder über dessen Schulter die Spielwelt sehen kann. 36, 49, 55, 84

hedonische Qualität Bezeichnet, welche Emotionen ein Tool hervorruft.[Mül10] 37

Lexer Ist die Abkürzung für lexikalischer Scanner. Dieser zerteilt einen Text in Token. [al21b] 12

Mann-Whitney-U Test Bestimmt, ob es zwischen zwei Gruppen einen statistisch signifikanten Unterschied gibt, ohne dass die Daten normalverteilt sein müssen. 57, 65, 67, 69

McCabe Komplexität Eine Metrik, um die Komplexität von Quellcodes anzugeben. 2, 40

pragmatische Qualität Bezeichnet, wie nützlich respektive praktisch in der Nutzung, ein Tool ist.[Mül10] 37

Prefab Ein Unity-Objekt, welches GUI-Elemente enthält und mithilfe von C# Skripten beliebig oft in Szenen eingefügt werden kann. 20

Rich-Text Besteht aus XML-Tags. Diese XML-Tags enthalten dann beispielsweise Farbcodes wie $< color = \#005500 > DarkGreen < /color >$. Der Text *Dark Green* wird damit in dunkel Grün dargestellt. [13] 9, 10, 12, 30, 31

Scipy Ein Framework für die Programmiersprache Python, mit dessen Hilfe u. a. statistische Berechnungen durchgeführt werden können. 57

SEE-Action Sind Aktionen, die ein Nutzer innerhalb von SEE ausführen kann. Hierzu zählt etwa das Löschen eines Gebäudes aus der Software-Stadt. 23

Syntax-Highlighting Die farbliche Hervorhebung bestimmter Schlüsselwörter einer Programmiersprache im Quellcode. 2, 9, 11, 12, 19, 20, 22, 30, 76

Token Ist ein atomarer Teil des zerteilten Quelltextes. [al14] 12

Viewport Der Bereich, in dem der Inhalt angezeigt wird. 20

[13] Eine komplette Einführung in die möglichen Formate mit Rich-Text in TMP gibt es hier: `http://digitalnativestudios.com/textmeshpro/docs/rich-text/` Hinweis: Das XML Beispiel stammt aus der zuvor genannten Quelle

Kapitel B AKRONYME

CRDT Ein Ansatz, um zwei oder mehr Versionen eines Textes zusammen zuführen. 15–19, 22, 24–26, 28–31, 76, 78

LOC Lines of Code, eine Metrik, die die Zeilenanzahl einer Klasse angibt. 2, 40–42, 45, 58, 59, 84

SEE Eine Software zur Visualisierung von Quellcode. 1–6, 8, 11, 12, 16, 20, 23, 33–37, 39, 41–44, 46–49, 54–72, 74, 75, 78, 84

SUS Ein validierter Fragebogen zur Erfassung der Benutzbarkeit von Software. 37–39, 49, 62–65, 75, 85

TMP Ein Framework zur Ein- und Ausgabe von Texten in Unity. 9, 12, 19, 27, 80

LITERATUR

[al14] Matthiasrella et al. *Token (Übersetzerbau) – Wikipedia*. https://de.wikipedia.org/wiki/Token_(%C3%9Cbersetzerbau). (Accessed on 01/29/2022). Juni 14.

[al21a] Moritz Blecker et al. *Projekt SEE - Software-Entwicklung im 21. Jahrhundert*. (Accessed on 01/21/2022). Mai 2021.

[al21b] Schewek et al. *Tokenizer – Wikipedia*. https://de.wikipedia.org/wiki/Tokenizer. (Accessed on 01/29/2022). Juli 2021.

[Bar] Chong-ho Yu Barbara Ohlund. *Threats to validity of Research Design*. https://web.pdx.edu/~stipakb/download/PA555/ResearchDesign.html. (Accessed on 01/21/2022).

[Ber16] SAP Bernard Rummel. *System Usability Scale – jetzt auch auf Deutsch. | SAP Blogs*. https://blogs.sap.com/2016/02/01/system-usability-scale-jetzt-auch-auf-deutsch/. (Accessed on 11/11/2021). Feb. 2016.

[Bha20] Pritha Bhandari. *External Validity | Types, Threats & Examples*. https://www.scribbr.com/methodology/external-validity/. (Accessed on 01/30/2022). Mai 2020.

[Che17] Rudi Chen. *A simple approach to building a real-time collaborative text editor - Digital Freepen*. https://digitalfreepen.com/2017/10/06/simple-real-time-collaborative-text-editor.html. (Accessed on 07/04/2021). Okt. 2017.

[Fra] unbekannt Fraunhofer Institut. *Weitere Usability-Fragebögen | Lab-IoT*. https://websites.fraunhofer.de/Lab-IoT/?page_id=1166. (Accessed on 11/11/2021).

[Kle18] Martin Kleppmann. *CRDTs and the Quest for Distributed Consistency - YouTube*. https://www.youtube.com/watch?v=B5NULPSiOGw. (Accessed on 07/13/2021). Okt. 2018.

[Kos19] Rainer Koschke. *Vorlesung Softwareprojekt 1 - Entwurfsmuster*. (Accessed on 01/21/2022). 2019.

[Kos20] Rainer Koschke. *Auge in Auge mit Ihrer Softwarearchitektur*. (Accessed on 10/10/2021). 2020.

[Mar19] Jack Mariani. *c# - Receive any Keyboard input and use with Switch statement on Unity - Stack Overflow*. https://stackoverflow.com/questions/56373604/receive-any-keyboard-input-and-use-with-switch-statement-on-unity/56373753. (Accessed on 08/19/2021). Mai 2019.

[Mül10] Julia Müller. *Das Geheimnis attraktiver Produkte - und wie man Attraktivität messen kann - Usabilityblog.de*. https://www.usabilityblog.de/das-geheimnis-attraktiver-produkte-und-wie-man-attraktivitat-messen-kann/#:~:text=Ein%20Produkt%20besitzt%20pragmatische%20Qualit%C3%A4t,auf%20Usability%20im%20eigentlichen%20Sinne.&text=Die%20wahrgenommene%20hedonische%20Qualit%C3%A4t%20erf%C3%BCllt%20die%20Bed%C3%BCrfnisse%20nach%20Neugier%20und%20sozialem%20Vergleich.. (Accessed on 01/30/2022). Feb. 2010.

[Ryt14] Ryte. *Was ist der System Usability Scale? – Ryte Wiki*. https://de.ryte.com/wiki/System_Usability_Scale#Der_SUS_Score. (Accessed on 01/20/2022). Sep. 2014.

[Sta] Statista. *Signifikanz | Statista*. https://de.statista.com/statistik/lexikon/definition/122/signifikanz/. (Accessed on 01/10/2022).

ABBILDUNGSVERZEICHNIS

Kapitel C ANHANG

- Ein USB-Stick mit Folgenden Inhalt:
  - SEE.exe — Eine auf Windowsrechnern lauffähige Version von SEE. Diese wurde für die Evaluation verwendet.
  - Anleitung.pdf — Die Präsentationsfolien der Anleitung für die Teilnehmer.
  - Auswertung.xlsx — Die Rohdaten der Evaluation. Inklusive der Diagramme. (Exklusive der Boxplots.)
  - Python-Auswertung — Das Script zur Auswertung der Signifikanz und Erstellung der Boxplots. (Inklusive CSV-Dateien und Boxplots als PDF.)
  - Korrektheit — Ein Ordner mit allen Texten und den Ergebnissen der Auswertung.
  - Fragebogen — Enthält alle Fragebogenseiten als PDF.
  - Bilder — Alle Bilder aus dieser Bachelorarbeit in Originalauflösung.